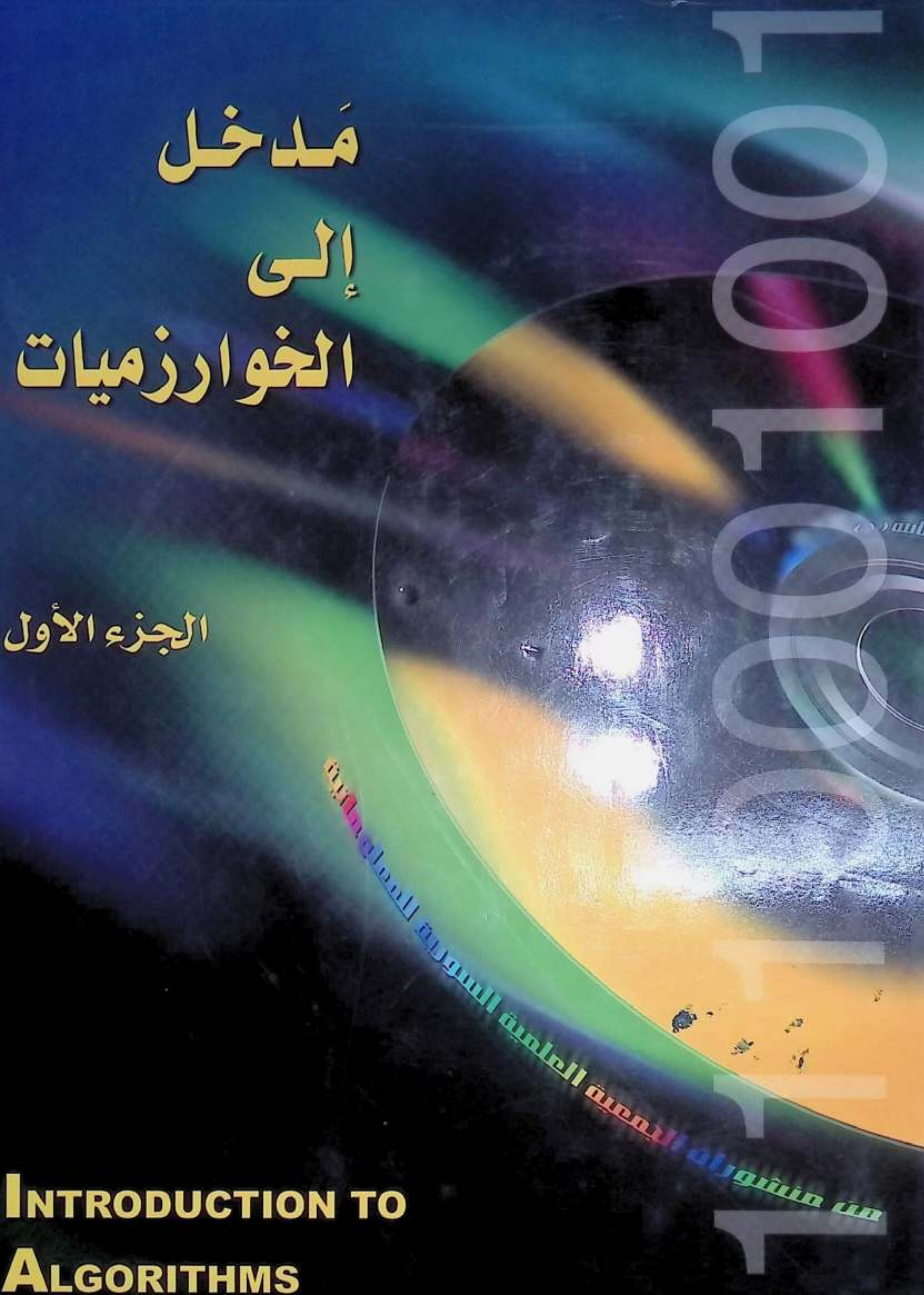


مَدخل
إلى
الخوارزميات
الجزء الأول
من منشورات الجمعية العلمية السورية للمعلوماتية
INTRODUCTION TO
ALGORITHMS

مَدْخل إلى الخوارزميات

Introduction to Algorithms

Third Edition

الجزء الأول

مَدْخل إلى الخوارزميات

الإصدار الثالث

تأليف

Thomas H. Cormen

Charles E. Leiserson

Ronald L. Rivest

Clifford Stein

ترجمة

د. عُلى أبو عمشة — د. محمد سعيد الدسوقي

د. أميمة الدكاك — د. ندى غنيم

د. كمال قمر — د. غيداء ربداوي

د. رضوان قسطنطين

مطبوعات الجمعية العلمية السورية للمعلوماتية

عنوان الكتاب الأصلي

Introduction to Algorithms

THIRD EDITION

Thomas H. Cormen
Charles E. Leiserson
Ronald L. Rivest
Clifford Stein

The MIT Press
Cambridge, Massachusetts   London, England

نظرة سريعة إلى محتوى الجزء الأول

الباب الرابع تقنيات متقدمة في التصميم والتحليل

الباب الخامس بنى المعطيات المتقدمة

نظرة سريعة إلى محتوى الجزء الثاني

الباب السادس خوارزميات البيانات

الباب السابع مواضيع مختارة

الباب الثامن الملاحق: معارف رياضية أساسية

محتوى الجزء الأول

الباب الثاني الفرز وإحصائيات الترتيب

الباب الثالث بنى المعطيات

الباب الرابع تقنيات متقدمة في التصميم والتحليل

الباب الخامس بنى المعطيات المتقدمة

كلمة الجمعية العلمية السورية للمعلوماتية

عزيزي القارئ

تضع الجمعيةُ العلميةُ السوريةُ للمعلوماتية اليوم بين يديك، وبعد طولُ انتظارٍ هذا الكتاب "مَدْخل إلى الخوارزميات"، الذي عَمِلَ على نَقْله إلى العربية مجموعةٌ متخصِّصةٌ من الباحثين والمدرِّسين في علوم الحاسوب عمومًا، والخوارزميات خصوصًا. ويأتي هذا الكتابُ في إطار جهودِ الجمعية المستمرة لإغناء المكتبة العربية بكتبٍ تخصصيةٍ في المعلوماتية بلغةٍ عربيةٍ سليمةٍ ومعاصرة تساعد القارئَ العربيَّ على الحصول على أكثر العلوم معاصرةً بلغته الأم.

وقد قامت لجنةُ التأليف والترجمة والنشر في الجمعية منذ العام 2000، بترجمةِ مجموعةٍ من كتب المعلوماتية المتميِّزة، التي يعدُّ بعضُها مرجعًا أساسيًّا لطلاب الجامعات. نذكر منها: "أسس لغات البرمجة" المنشور عام 2000، و"هندسة البرمجيات منهج للممارس" (في جزأين) عام 2001، و"الذكاء الصنعي" عام 2004، و"مفاهيم نظم التشغيل" (في جزأين) عام 2005، و"التعمية التطبيقية" و"اتصالات المعطيات والحواسيب" عام 2006، إضافة إلى منشورات أخرى لا تقلّ عنها أهميةً مثل: معجم مصطلحات المعلوماتية الذي نُشِرَ في العام 2000، والذي يقع الآن في صُلْبِ مشروعٍ عربيٍّ بالتعاون مع الاتحاد العالمي للاتصالات لإصدار نسخةٍ موسَّعةٍ ومُحَدَّثةٍ منه.

يعالِج هذا الكتابُ أحدَ أهمِّ الأسس في علوم الحاسوب، ألا وهو بنى المعطيات والخوارزميات. ولا يقتصر هذا الكتابُ – كغيره من كتبِ الخوارزميات العديدة – على كونه "كتاب وَصْفات" يبيِّن أهمَّ ما استقرَّت عليه الدراساتُ والبحوث في هذا المجال، بل يأخذ بيدِ القارئِ خطوةً خطوة؛ فيشرح كلَّ مسألةٍ بالتفصيل، ثم يَعرض مجموعةً من الحلول ويقارن بينها مستعينًا بأدواتِ الرياضيات المتقطِّعة التي لا غِنَى عنها للوصول إلى فهمٍ عميقٍ لهذه الحلول. ويرمي هذا الأسلوبُ إلى تطويرِ قدرةِ الدارس تدريجيًّا على المقارنة والنقد واختيار وتصميم الحلول الفضلى للمسائل التطبيقية التي قد تعترضه.

إن النسخةَ الأصليةَ من هذا الكتاب "Introduction to Algorithms" هي كتابٌ مرجعيٌّ من منشورات دار نشر معهد ماساتشوستس للتقانة MIT Press. وهو مرجعٌ تدريسيٌّ معتمَدٌ في معظم جامعات العالَم، ويعدُّ من أكثر الكتبِ المرجعيةِ مَبيعًا؛ فقد بلغ عددُ النسخ المبيعة منه حتى آب 2011 نصف مليون نسخة[1]، وذلك منذ إصداره الأول في العام 1990. وهو إلى ذلك كتابٌ شاملٌ يقدِّم العونَ للطالب من بداية دراسته الجامعية وحتى دراساته العليا. ومما يدلُّ على كبيرِ أهميةِ هذا الكتاب وعلوِّ شأنه في بابِهِ أنه وَرَدَ في مَراجعِ ما يزيد على خمسةِ آلاف بحثٍ باعتباره أحدَ المراجعِ الأساسيةِ في الخوارزميات، إضافةً إلى أنه مرجعٌ لا يُستغنى عنه في جميع مجالات علوم الحاسوب[2].

---

[1] http://web.mit.edu/newsoffice/2011/introduction-to-algorithms-500k-0810.html

[2] http://citeseerx.ist.psu.edu

يتجاوز عددُ صفحات الكتاب الأصلي 1200 صفحة، ولهذا السبب رأى فريقُ التعريب إصدارَ النسخة العربية في جزأين (متقاربَيْن في عدد الصفحات) لتسهيل استعماله. يتضمَّن الجزءُ الأولُ أساسياتِ تحليل الخوارزميات، وخوارزمياتِ الفرز، وبنى المعطيات الأساسية، وبنى معطيات متقدمة، وبعضَ الطرق المتقدمة في حلِّ المسائل. ويضمُّ الجزءُ الثاني - الذي سيصدر لاحقًا - خوارزمياتِ نظرية البيان، إضافةً إلى تسعةِ فصولٍ تعالِج مواضيعَ مختارة ذات أهميةٍ كبيرة. ويختتم الكتابُ بمجموعةٍ من الملاحق في مواضيعَ رياضيةٍ ذات صلةٍ وثيقةٍ بالخوارزميات، ثم قائمة مراجع غنيةٍ تزيد على 350 مرجعًا، ومسارِدَ تبيِّن المصطلحاتِ باللغتين العربية والإنكليزية.

مؤلِّفو الكتابِ بدءًا من إصداره الثاني أربعة: اثنان منهم مدرِّسان في معهد التقانة في ماساتشوستس، وهما Ron Rivest و Charles E. Leiserson، والآخران من طلاب الدراسات العليا اللامعين في المعهد في الثمانينيات، وهما Thomas H. Cormen و Clifford Stein. وجميعُهم الآن من العلماء المرموقين، ولهم إسهاماتٌ في العديد من مجالات علومِ الحاسوب. فعلى سبيل المثال، قدَّم Rivest العديدَ من الإنجازات في مجال علم التعمية، وهو أحدُ مطوِّري خوارزمية RSA الشهيرة. ولزميله Leiserson إسهاماتٌ كثيرةٌ في الحوسبة التفرُّعية والحوسبة الموزَّعة، وهو أحدُ رُوَّاد تطوير نظرية VLSI، ومصمِّمُ شبكةِ الوصل fat tree المستعمَلة في الحواسيب الفائقة. أما Cormen، فهو مدرِّسٌ وكاتبٌ بارع، تعدَّدت إنجازاتُه في مجال التعليم الجامعي، وهو رئيسُ قسم برنامج الكتابة في جامعة دارتموث. وأما Stein، فقد أسهم في تأليف كتاب "مَدْخل إلى الخوارزميات" بدءًا من إصداره الثاني، وهو مختصٌّ في بحوثِ العمليات، ويرأس حاليًّا قسمَ الهندسة الصناعية وبحوث العمليات في جامعة كولومبيا، وقد أَغْنَى الكتابَ بالعديد من الأمثلة التطبيقية.

أما فريقُ التعريب فقد ضمَّ نخبةً من خيرة الباحثين والمدرِّسين ذوي الخبرة الطويلة في المعهد العالي للعلوم التطبيقية والتكنولوجيا وفي جامعة دمشق، وهم: د. عُلى أبو عمشة، و د. محمد سعيد دسوقي، و د. أميمة الدكاك، و د. ندى غنيم، و د. كمال قمر، و د. غيداء ربداوي، و د. رضوان قسطنطين. وقد قاموا معًا بعملٍ تعاونيٍّ ضخم - ضمن فريقٍ واحدٍ متكامل - تَمثَّل في تعريبِ فصولِ الكتاب، كلٌّ منهم فيما هو أقرب إلى خبرته واختصاصه، ثم تداولوا مراجعةَ هذه الفصول فيما بينهم. وبعد ذلك، قام الأستاذ مروان البوّاب مشكورًا بالمراجعة اللغوية وبإدخال التعديلات الناتجة عنها. وكانت د. عُلى أبو عمشة هي المسؤولةَ عن تنسيقِ العمل بين أعضاء الفريق، وعن تعريبِ المصطلحات العلمية وتوحيدِها، وأَجْرَتْ لهذا الغرض سلسلةً من المناقشات مع زملائها. وأخيرًا قام د. رضوان قسطنطين بعملٍ مهمٍّ يُعَدُّ بحقٍّ تتويجًا لهذا الكتاب؛ فنسَّق نصوصَه، وضبط معادلاتِه، ورتَّب مقاطعَه البرمجية، وأخرجه بحلَّةٍ أنيقةٍ تُماثِل النسخةَ الأصلية له.

ويطيب لنا أن نشكر أفرادَ أسرةِ العمل كافةً على الجهد الكبير الذي بذلوه في التعريب والمراجعة العلمية واللغوية. ونخصُّ بالشكر د. عُلى أبو عمشة - التي كانت المحرِّكَ الأوَّلَ للانطلاق بهذا العمل - على جهودها في التنسيق، وفي العمل على توحيدِ المصطلحات في هذا الكتاب الضخم. ونوجِّه شكرًا خاصًّا أيضًا إلى د. رضوان قسطنطين على عنايته الكبيرة في إخراج النص وإدخال الأشكال والمعادلات الكثيرة لتظهر بأسلوب موحَّدٍ في كامل النص، وعلى جهوده للمحافظة على توحيد المصطلحات. ونشير هنا إلى أن د. قسطنطين - إضافةً إلى أنه تَدارَكَ الهفواتِ التي عثر عليها في

أثناء إخراجه للكتاب – قام بتصحيح الأخطاءِ المنشورةِ على موقع الكتاب[3] حتى تاريخ الانتهاء من إعداد نسخته المعرَّبة.

والشكر موصولٌ كذلك إلى الأستاذ مروان البوَّاب على جهوده في مراجعة هذا الكتاب، التي شملت الجوانبَ اللغويةَ والعلميةَ أيضًا، بعنايةٍ ودقةٍ كبيرتين. وقد كانت له ملاحظاتٌ دقيقةٌ فيما يخصُّ تعريبَ العديد من المصطلحات.

وفي الختام نتوجَّه بالشكر الجزيل إلى الأستاذ الدكتور راكان رزوق، رئيس مجلس إدارة الجمعية العلمية السورية للمعلوماتية على دعمه للجنة الترجمة والتأليف والنشر، وتشجيعه على أن يرى هذا الكتاب النور.

وأخيرًا نأمُل أن نكون بكتابنا هذا قد وُفِّقنا في وضع ما يساعد على فهمِ مواضيع اتصالات المعطيات والحواسيب بين يدي القارئ العربي. ونرجو أن يَصدر الجزءُ الثاني منه في القريب العاجل.

والله ولي التوفيق.

الأستاذ الدكتور وائل معــلا

رئيس لجنة الترجمة والتأليف والنشر

في الجمعية العلمية السورية للمعلوماتية

---

[3] http://mitpress.mit.edu/algorithms/#bugs

كلمة فريق التعريب

زميلنا المدرِّس

عزيزنا الطالب

نقدِّم لك الجزءَ الأولَ من النسخةِ المعرَّبة للكتاب المرجعيِّ الشهير "Introduction to Algorithms"، وبإذن الله سيصدر الجزءُ الثاني منه بعد مدةٍ غير طويلة.

إن هذا العملَ ثمرةُ جهودِ فريقنا التي امتدَّت عدةَ سنوات. ولن يخفى عليك، أيها القارئ العزيز، عندما تبدأ بقراءةِ صفحاتِ هذا الكتاب مدى الجهود المبذولة من مؤلِّفي الكتاب لجعله شاملاً وواضحًا ودقيقًا، ومن فريقنا الذي بذل قصارى جهده ليقدِّم لك عملاً علميًّا رائعًا بلغةٍ عربيةٍ سليمةٍ وبسيطة، ولينقل الأفكارَ وحتى أسلوب المؤلِّفين بأقصى قدرٍ من الأمانة، راجين بذلك أن نتيح لقارئنا العربي الاطلاع على منشوراتٍ قيِّمةٍ بلغته الأم، عسى أن تصبح أمتنا العربية من جديدٍ منتِجةً للعلوم ومطوِّرةً لها.

تَعود جذورُ هذا الكتاب إلى منتصفِ السبعينيات، وكان قوامُهُ وقتئذٍ محاضراتٍ في الخوارزميات أُلقيت في معهد MIT. وفي منتصف الثمانينيات شَجَّعت براعةُ Cormen في الكتابة العلمية أساتذتَه آنذاك على خوضِ مغامرةِ تأليفِ كتابٍ صَدَرَ في العام 1990، وزاد عددُ صفحاته على 1000 صفحة، وصار مع مرور الأيام المرجعَ الأساسيَّ للخوارزميات في العديد من جامعات العالَم. وقد بدأتْ صلةُ فريقنا بالكتاب باعتماده مرجعًا أساسيًّا في تدريس مقرَّر الخوارزميات في المعهد العالي للعلوم التطبيقية والتكنولوجيا وفي جامعة دمشق، وكنا نستمتع بعمقِهِ ودِقَّتِهِ واستخدامه للرياضيات بالقدر المناسب لتعميق مفاهيم الخوارزميّات. ونشأتْ بذلك فكرةُ نَقْلِهِ إلى العربية لنشارك به طلابَنا وكلَّ مَن يَعُوقُهُ حاجزُ اللغة عن الاستفادة من هذا الكتاب. وقد قَبِلَت الجمعيةُ العلميةُ السوريةُ للمعلوماتية مشكورةً اعتمادَ هذا المشروع ضمن جهودها لإغناء المكتبة العربية التقنية بالكتب القيِّمة.

وقد تكوَّن فريقُنا لإنجاز مهمةِ تعريبِ هذا الكتاب في العام 2007. وكان أول ما بدأنا به هو تعريب مَسْرَدِ المصطلحاتِ لضمان توحيدها في جميع فصول الكتاب، على الرغم من تعدُّد مترجمي هذه الفصول. وقد اعتمدنا في ترجمة هذه المصطلحات في المقام الأول على ما وَرَدَ في معجم مصطلحات المعلوماتية الصادر عن الجمعية عام 2000، واجتهدنا – في العديد من الاجتماعات وبالتعاون مع المدقِّق اللغوي – في ترجمةِ المصطلحاتِ التي لم تَرِدْ في المعجم. وعلى الرغم من كلِّ الجهود المبذولة في هذا الصدد، فقد يكون هناك أكثرُ من مقابلٍ بالعربية لبعضِ المصطلحات الإنكليزية، إلا أننا نعتقد أنه لن يكون لهذا كبيرُ أثرٍ على فهمِ القارئ لمضمون الكتاب، وذلك بفضل ما تتمتَّع به فصولُ الكتاب من استقلاليةٍ فيما بينها.

وتجدر الإشارة إلى أن عَمَلَنا بدأ بتعريبِ الإصدار الثاني لهذا الكتاب، وبعد أن قاربنا على الانتهاء من الجزء الأول منه، عَلِمْنا بصدورِ الإصدار الثالث، وكان ذلك في آب 2009. وقد احتوى هذا الإصدارُ الجديدُ تحديثاتٍ كثيرةً جعَلَته

أكثرَ قُربًا من القارئ، إضافةً إلى تدارُك بعضِ النقاط التي كانت معالَجةً بطريقةٍ مختلفةٍ عمَّا كان دارجًا في كتبِ الخوارزميات الأخرى، وأُغْنِيَت مَلاحِقُهُ بحيث لا يحتاج القارئُ إلى العودةِ إلى كتب الرياضيات لاستذكار المفاهيم اللازمة لفهم النصّ. فلم يكن بوسع فريقنا أن يَحرِم النسخةَ العربيةَ من هذه التحديثات، لاسيَّما أن مؤلِّفي هذا الكتاب يَعملون سنواتٍ عدَّة قبل إطلاق إصدارٍ جديدٍ له، فاتخذنا قرارًا صعبًا بمراجعةِ جميعِ الفصول المعرَّبة وجَعْلِها مطابقةً للإصدار الثالث.

يعالج الكتابُ الخوارزمياتِ التي تقع في الواقع في صُلْب علم الحاسوب وتطبيقاته، فيبدأ بالتعريف بالخوارزميات وبجوانب تحليلها المتعدِّدة، ثم يقدِّم في كلِّ فصل: بنيةَ معطياتٍ مع الخوارزميات المتعلقة بها، أو تقنيةَ تصميم، أو مجالاً تطبيقيًّا، أو موضوعًا ذا صلةٍ بعنوان الفصل. وقد راعى المؤلِّفون أن تكون الفصولُ مستقلةً فيما بينها قدرَ الإمكان، لتكونَ قراءةُ أيِّ فصلٍ سهلةً وواضحةً دون الحاجة الكبيرة إلى الاطلاع على الفصول التي سبقته، ما عدا، ربما، فصول الباب الأول التي تُعْنَى بالأساسيات في تحليل الخوارزميات، أو الملاحق لتلافي الثغرات في الأساس الرياضي للقارئ.

يقع الكتابُ في خمسةٍ وثلاثين فصلاً موزَّعة على سبعةِ أبواب. يضمُّ الجزءُ الأولُ من النسخةِ المعرَّبة خمسةَ أبواب:

- يزوِّد البابُ الأولُ القارئَ بالسياقِ والأدواتِ اللازمةِ للمضيِّ قُدُمًا في الكتاب. فهو يعالِج مبادئَ تصميم الخوارزميات وتحليلها. ويضمُّ مقدِّمةً سهلةً في تَوصيف الخوارزميات والأدوات اللازمةِ لذلك، ثم يُعرِّف بعضَ استراتيجيات التصميم المهمة المستعمَلة لاحقًا في الكتاب، وخاصةً استراتيجية "فرِّق-تسد".
- يعالِج البابُ الثاني مسألةَ فرزِ الأعداد التي تقع في صُلْبِ معظمِ التطبيقات المحوسبة، ويقدِّم خوارزمياتٍ عديدةً لحلِّ هذه المسألة، ويدرس أمثليَّتها. ويعالِج الفصلُ الأخيرُ من هذا الباب ما يُعْرَف بإحصائيات الترتيب.
- يَعرِض البابُ الثالثُ بعضَ بنى المعطيات والتقنيات الأساسية لتمثيل مجموعات المعطيات الديناميكية المنتهية، وكيفية التعامل معها حاسوبيًّا من استفسارٍ وتعديلٍ وتنظيم. تَشمل بنى المعطيات المدروسة في هذا الباب البنى البسيطة مثل: المِكْدَس، والرَّتَل، والقائمة المترابطة، والشجرة ذات الجذر. ثم ينتقل إلى جداول التَّلْبِيد والأشجار الثنائية والأنواع المُغْناة منها.
- يَدرس البابُ الرابعُ ثلاثَ تقنياتٍ هي: البرمجة الديناميكية، والخوارزميات الشرهة، والتحليل المخمَّد. وتضاف هذه التقنياتُ إلى تلك التي يقدِّمها البابُ الأول، وهي تُستعمَل بكثرةٍ في تصميم الخوارزميات الفعالة وتحليلها.
- البابُ الخامسُ مخصَّصٌ لدراسةِ بعضِ بنى المعطيات المتقدِّمة التي تدعم العملياتِ على المجموعات الديناميكية بفعاليةٍ أكبر من البنى المعروضة في الباب الثالث، لكنها أكثر تعقيدًا، إذ تَستعمل بعضُ البنى المدروسة بكثافةٍ تقنياتِ التحليل المخمَّد التي يعالجها البابُ الرابع. أما البنى المدروسةُ في هذا الباب، فهي: الأشجارُ المعمَّمة، والكوماتُ القابلةُ للدمج، وكوماتُ فيبوناتشي، وبنيةُ المعطيات العَوْدِيَّة المعروفة باسم شجرة van Emde Boas. وأخيرًا يعالِج هذا البابُ بنيةَ معطياتٍ خاصةٍ تُعْرَف بالمجموعات المنفصلة.

ويضمُّ الجزءُ الثاني بابَيْن وأربعةَ ملاحق:

- يعالِج البابُ السادسُ البياناتِ graphs وأهمَّ ما يتعلق بها من خوارزمياتٍ لها تطبيقاتٌ واسعةٌ في علوم الحاسوب والاتصالات. فيَعرض بعمقٍ خوارزمياتِ البحث داخلَ بيانٍ مع تطبيقاتٍ لها، وخوارزمياتِ البحث عن أشجار المسح الصغرى، وخوارزمياتِ أقصر المسارات بأنماطها المختلفة. وينتهي البابُ بدراسةِ خوارزمياتٍ تتعلَّق بحسابِ التدفُّق الأعظمي داخل بيان.
- يَجمع البابُ السابع – وهو الأخير – مجموعةً من المواضيع المختارة التي لا تقلّ أهميةً عن مواضيع الأبواب السابقة، ولكنها تُعَدُّ أكثرَ تقنيةً أو تعقيدًا. تتميَّز فصولُ هذا الباب بأنها – في جلِّها – مستقلةٌ بعضها عن بعض، وتعالِج مسائلَ محدَّدة يُفيد منها المتخصِّصون في مجالاتٍ مختلفة؛ فهناك مثلاً فصلٌ يُعْنَى بالحوسبةِ التفرُّعيةِ المتعدِّدةِ النياسب، وآخرُ بمطابقةِ متتالياتِ المحارف. وهناك فصولٌ تعالِج مواضيعَ وبنًى رياضيةٍ واسعةِ التطبيقات مثل: المصفوفات، وتحويلات فورييه، وخوارزميات نظرية الأعداد ذات الأهمية الكبيرة في مجال التعمية، وخوارزميات الهندسة المحوسبة. ويضمُّ البابُ أيضًا عدةَ فصولٍ تُعْنَى بحلِّ مسائلِ الأمثَلة optimization مثل: البرمجة الخطية، ونظرية تعقيد المسائل، وخوارزميات التقريب.
- تقدِّم الملاحقُ تذكرةً لكلِّ ما يحتاج إليه القارئُ من معارفَ ذاتِ طابعٍ رياضيّ، مثل: مجاميع السلاسل العددية، والبنى الرياضية مثل: المجموعات، والدوالّ، والعلاقات، والبيانات، والمصفوفات. وتذكِّر أيضًا بطرائق العد، ونظرية الاحتمالات.

ولكي يَسْهُلَ على القارئ الإفادة من هذا الكتاب القيِّم على أكملِ وجه، سيُزوَّد جزؤه الثاني بفهرسٍ ومَسْرَدٍ ألفبائيٍّ بالمصطلحات العربية ومقابلاتها الإنكليزية، وبمسْرَدٍ آخرَ بالمصطلحات الإنكليزية ومقابلاتها العربية.

نرجو أن نكون قد قدَّمنا بعملنا هذا الفائدةَ للقارئ العربي، وأسهمنا في تطوير وطننا وأمتنا العربية. ونأسف لما يمكن أن يكون قد غاب عنَّا من هفواتٍ وأخطاء، ونشكر سلفًا كلَّ مَن يُنبِّهنا عليها لتلافيها في طبعاتٍ قادمة.

والله وليّ التوفيق.

فريق التعريب

مقدمة

وُجِدَت الخوارزميات قبل وجود الحواسيب. واليوم ومع وجود الحواسيب، أصبح لدينا المزيد من الخوارزميات، وأصبحت الخوارزميات من صميم الحوسبة.

يقدم هذا الكتاب مدخلاً شاملاً لدراسة الخوارزميات الحاسوبية الحديثة، فهو يعرض عدة خوارزميات ويشملها بعمق، ومع ذلك فهو يُبقي تصميمَها وتحليلَها في متناول القراء على اختلاف مستوياتهم. حاولنا إبقاء الشروح بسيطة دون التضحية بعمق الشمول أو بالدقة الرياضية.

يعرض كل فصلٍ خوارزميةً، أو تقنيةَ تصميمٍ، أو مجالاً تطبيقيًّا، أو موضوعًا ذا صلة. تُشرَح الخوارزميات بالعربية وبشبهِ رماز مصمَّمٍ بحيث يتمكن أي شخص ملمٍّ بالبرمجة من قراءته. يتضمن هذا الكتاب 244 شكلاً — يحتوي العديد منها على عدة أجزاء — توضح كيف تعمل الخوارزميات. ولما كنا نشدد على *الفاعلية* باعتبارها معيارًا تصميميًّا، فقد ضمنّا تحليلات دقيقة لأزمان تنفيذ جميع خوارزمياتنا.

هذا النص معدٌّ أصلاً للاستعمال في المقررات المتعلقة بالخوارزميات وبنى المعطيات في المرحلة الجامعية وفي الدراسات العليا. ولما كان هذا الكتاب يناقش القضايا الهندسية في تصميم الخوارزميات، إضافةً إلى الجوانب الرياضية، فهو ملائم أيضًا لتعلُّم المختصين التقنيين ذاتيًّا.

في هذه الإصدار الثالث، قمنا بتحديث كامل الكتاب مرة أخرى. تشمل هذه التغييرات طيفًا واسعًا، من إدراج فصولٍ جديدة، وتعديل لشبه الرماز، واستخدام أسلوب كتابة موجه للقارئ.

إلى المدرِّس:

لقد صممنا هذا الكتاب ليكون متعدد الجوانب وكاملاً في الوقت نفسه. ستجده مفيدًا لمقررات متنوعة، ابتداءً من مقررات في بنى المعطيات في المرحلة الجامعية وحتى دروس في الخوارزميات للدراسات العليا. ولما كنا قد قدمنا موادَ أكثر بكثير مما يمكن أن يستوعبه مقرر نموذجي في فصل واحد، فيمكنك أن تعتبر هذا الكتاب مائدة مفتوحة منوعة يمكنك أن تنتقي وتختار منها أفضل مادة تدعم المقرر الذي ترغب في تدريسه.

ستجد أنه من السهل تنظيم مقررك المتعلق بالفصول التي تحتاج إليها فقط، فقد جعلنا الفصول مستقلاً بعضها عن بعض نسبيًّا، فلا تقلق بشأن الارتباط غير المتوقع وغير الضروري لفصل بآخر. يعرضُ كلُّ فصلٍ

المادةَ بتدرج من الأسهل إلى الأصعب، وتشير حدود المقاطع إلى نقاط التوقف الطبيعية. قد تستخدم في أحد مقررات المرحلة الجامعية، المقاطع الأولى من الفصل، في حين قد يغطي مقرر في الدراسات العليا الفصل بأكمله.

ضمنًا في الكتاب 957 تمرينًا و 158 مسألة. ينتهي كلُّ مقطع بتمارين، وكل فصل بمسائل. تكون التمارين عادةً أسئلةً قصيرة تختبر تمكّن الطالب من المادة. بعض هذه التمارين بسيط للتحقق الذاتي من الأفكار، في حين أن بعضها الآخر أكثر عمقًا ويناسب وظيفةً منْزلية. تمثل المسائل دراسةَ حالةٍ أكثر تفصيلاً، وغالبًا ما تقدم مادةً جديدة؛ وتتألف غالبًا من عدة أسئلة تقود الطالب عبر الخطوات اللازمة للوصول إلى حل.

انطلاقًا من خبرتنا في إصدارات سابقة من هذا الكتاب، وضعنا حلولاً لبعض المسائل والتمارين، لا لكلها، وجعلناها متاحة للعموم. يشير موقعنا /http://mitpress.mit.edu/algorithms إلى هذه الحلول. من المناسب أن تتفقد هذا الموقع لتتحقق من عدم تضمُّنه حلاً لتمرين أو مسألة تخطط لإعطائها وظيفةً منْزلية. ونتوقع أن تكبر مجموعة الحلول التي نُحمِّلها على مر الوقت، لذلك قد تحتاج إلى تفقد الموقع في كل مرة تدرّس المادة.

وضعنا علامة (⋆) على المقاطع والتمارين التي تناسب طلاب الدراسات العليا أكثر من المرحلة الجامعية. ولا يعني وضع علامة النجمة على مقطع بأنه أكثر صعوبة من المقطع الذي ليس موسومًا بنجمة، ولكنه قد يتطلب فهمًا لرياضيات أكثر تقدمًا. كذلك، فقد تتطلب التمارين الموسومة بالنجمة خلفية أكثر عمقًا، أو إبداعًا أكثر من الوسطي.

إلى الطالب:

نأمل أن يزودك هذا الكتاب بمدخل ممتع في مجال الخوارزميات. حاولنا جعل جميع الخوارزميات مفهومة ومثيرة للاهتمام. لمساعدتك عندما تصادفك خوارزميات غير شائعة أو صعبة، وصَّفنا كل واحدة منها خطوة خطوة، وقدَّمنا أيضًا شرحًا دقيقًا للرياضيات الضرورية لفهم تحليل الخوارزميات. إذا كانت لديك معرفة بسيطة عن موضوع ما، ستجد الفصول منظمة بحيث يمكنك تصفح المقاطع التمهيدية والبدء سريعًا بمواد أكثر تقدمًا.

إن هذا الكتاب كبير، وسيغطي صفُّك على الأرجح جزءًا من مواده فقط. غير أننا حاولنا أن نجعله كتابًا مفيدًا لكَ الآن ككتاب مرجعي للمقرر، ولاحقًا في مهنتك كمرجع مكتبي رياضي أو كدليل هندسي.

ما هي المتطلبات التي تلزمك لقراءة هذا الكتاب؟

- أن تكون لديك بعض الخبرة البرمجية. وبالتحديد، يجب أن تكون قد استوعبت الإجراءات العودية recursive procedures، وبنى المعطيات البسيطة كالصفيفة array، واللوائح المترابطة linked lists.
- أن تكون لديك بعض البراعة فيما يتعلق بالبراهين الرياضية، وخاصة البراهين بالاستقراء الرياضي mathematical induction. تعتمد بعض أجزاء هذا الكتاب على بعض المعارف بحسابات التكامل الأولية. فيما عدا ذلك، يُعلِّمك البابان I و VIII من هذا الكتاب جميع التقانات الرياضية التي ستحتاج إليها.

لقد سمعنا طلبكم الواضح والصريح لتزويدكم بحلول المسائل والتمارين. يضع موقعنا http://mitpress.mit.edu/algorithms/ وصلات إلى حلول لبعض المسائل والتمارين. يمكنكم متى شئتم مقارنة حلولكم بحلولنا، لكن لا ترسلوا إلينا حلولكم.

إلى المختص:

إن الطيف الواسع للمواضيع الموجودة في هذا الكتاب يجعل منه دليلاً ممتازًا عن الخوارزميات. ولما كان كلُّ فصلٍ مستقلَّ المضمون تقريبًا، يمكنك أن تركز على أكثر المواضيع أهمية بالنسبة إليك.

إن أغلب الخوارزميات التي نتطرق إليها ذات فائدة عملية عظيمة. لذلك، نحن نناقش قضايا التنجيز ومسائل هندسية أخرى. نقدم غالبًا بدائل عملية لبعض الخوارزميات التي لها أهمية نظرية في المقام الأول.

إذا رغبت في تنجيز أيٍّ من الخوارزميات، ستجد أن ترجمة شبه الرماز pseudocode الذي كتبناه، إلى لغة البرمجة المفضلة لديك، هي مهمة مباشرة إلى حدٍّ ما. لقد صممنا شبه الرماز لعرض كل خوارزمية عرضًا واضحًا وموجزًا. ومن ثم، نحن لا نناقش قضايا معالجة الخطأ وقضايا هندسة البرمجيات الأخرى التي تتطلب افتراضات محددة حول البيئة البرمجية التي تستخدمها. نحاول عرض كل خوارزمية عرضًا بسيطًا ومباشرًا دون أن نسمح لخصوصيات لغة برمجة محددة أن تغطي جوهرها.

نحن نتفهم أنك إذا كنت تستخدم هذا الكتاب خارج نطاق أي مقرر، فربما لن تكون قادرًا على تدقيق حلولك للمسائل والتمارين ومقارنتها بالحلول التي قد يقدمها مدرس. يُوجد على موقعنا http://mitpress.mit.edu/algorithms/ وصلات إلى حلولٍ لبعض المسائل والتمارين، بحيث تتمكن من تدقيق عملك. يرجى عدم إرسال حلولكم لنا.

إلى زملائنا:

لقد وفرنا مراجع ومؤشرات شاملة على الأدبيات الحالية. ينتهي كل فصل بمجموعة من الملاحظات التي تقدم تفاصيل تاريخية ومراجع. غير أن ملاحظات الفصول لا تقدم دراسة مرجعية كاملة لمجال الخوارزميات كله. وقد يصعب التصديق أن قيود حجم الكتاب منعتنا من إضافة خوارزميات هامة عديدة.

رغم الأعداد الضخمة لطلبات الطلاب للحصول على حلول للمسائل والتمارين، فقد اخترنا سياسة عدم التزويد بمراجع لحل المسائل والتمارين، لئلا تسول للطلاب أنفسهم البحث عن حلٍّ للمسألة بدلاً من حلها بأنفسهم.

التغييرات على الإصدار الثالث:

ما الذي تغير بين الإصدار الثاني والثالث من هذا الكتاب؟ يعادل حجم التغييرات الحالية حجم التغييرات بين الإصدارين الأول والثاني. وكما ذكرنا عن التغييرات في الإصدار الثاني، قد تجد التغييرات في هذا الإصدار كثيرة

أو محدودة تبعاً لكيفية قراءتك للكتاب.

تبيِّن نظرة سريعة إلى الفهرس أن معظم فصول ومقاطع الإصدار الثاني موجود في الإصدار الثالث. قمنا بحذف فصلين ومقطع واحد، ولكننا أضفنا ثلاثة فصول جديدة ومقطعين، عدا الموجود في هذه الفصول الجديدة.

لقد حافظنا على التنظيم الهجين المعتمد في الإصدارين السابقين. فبدلاً من تنظيم الفصول وفق مواضيع (أو مجالات تطبيق) المسائل فقط أو وفق التقنيات فقط، يعتمد الكتاب مزيجًا من الاثنين معًا، فهو يتضمن فصولاً تعتمد التقنيات، مثل فرِّق-تَسُد divide-and-conquer، والبرمجة الديناميكية dynamic programming، والخوارزميات الشرهة greedy algorithms، والتحليل المخمَّد amortized analysis، وتعقيد المسائل NP-Completeness NP، وخوارزميات التقريب approximation algorithms. لكنه يتضمن أيضًا أجزاءً كاملة عن الفرز، وبنى المعطيات اللازمة للمجموعات الديناميكية، والخوارزميات المتعلقة بمسائل البيان graph problems. ونحن نعلم أنه على الرغم من أنك بحاجة إلى معرفة كيفية تطبيق التقنيات في التصميم والتحليل الخوارزمي، إلا أنه قلما تدلُّك المسائل على أكثر التقنيات طواعيةً لحلها.

نقدم فيما يلي ملخصًا لأهم التغييرات في الإصدار الثالث:

- أضفنا فصولاً جديدة عن أشجار van Emde Boas، والخوارزميات المتعددة النياسب multithreaded algorithms، وفصلنا المادة المتعلقة بأساسيات المصفوفات في فصلٍ ملحقٍ خاص.
- راجعنا الفصل المتعلق بالعودية recurrences ليشمل بصورة أوسع تقنية "فرِّق-تَسُد"، وبحيث تُطبَّق هذه التقنية لحل مسألتين في أول مقطعين فيه. يعرض المقطع الثاني من هذا الفصل خوارزمية Strassen لإيجاد جداء المصفوفات، نقلناها من الفصل المتعلق بعمليات المصفوفات.
- حذفنا فصلين كانا نادرًا ما يدرَّسان: الكومات الثنائية binomial heaps، وشبكات الفرز sorting networks. تظهر في هذا الإصدار فكرة أساسية من فصل شبكات الفرز، وهي مبدأ 0-1، وذلك ضمن المسألة 8-7 على شكل توطئة الفرز 0-1 في خوارزميات قارِن-بدِّل compare-exchange. ولم تَعُد معالجة كومات فيبوناتشي Fibonacci تعتمد على الكومات الثنائية باعتبارها متطلبًا سابقًا.
- راجعنا طريقة معالجتنا للبرمجة الديناميكية والخوارزميات الشرهة. تبدأ البرمجة الديناميكية الآن بمسألة أكثر إمتاعًا، وهي تقطيع القضبان rod cutting، بدلاً من مسألة جدولة خط التجميع assembly line الموجودة في الإصدار الثاني. إضافة إلى ذلك، ركزنا على الاستذكار أكثر مما فعلنا في الإصدار الثاني، وقدمنا فكرة بيان المسألة الجزئية على أنها طريقة لفهم زمن تنفيذ خوارزمية البرمجة الديناميكية. في مثالنا الافتتاحي عن الخوارزميات الشرهة، وهو مسألة اختيار النشاط activity-selection، نصل إلى الخوارزمية الشرهة مباشرة بطريقة أفضل مما كانت في الإصدار الثاني.

- تضمن الطريقة التي نحذف وفقها الآن عقدة من أشجار البحث الثنائية (التي تتضمن الأشجار الحمراء-السوداء) أن العقدة التي يطلب حذفها هي العقدة المحذوفة فعليًا. في الإصدارين السابقين، وفي بعض الحالات، كان من الممكن أن تُحذف عُقدةٌ ما، وتُنقل محتوياتها إلى العقدة التي تُمَرَّر إلى إجراء الحذف. مع طريقتنا الجديدة لحذف العقد، إذا كانت هناك مكونات أخرى من البرنامج تحافظ على مؤشرات إلى عقد في الشجرة – فلن ينتهي بها الأمر مع مؤشرات قديمة إلى عقدٍ حُذِفت.
- إن المادة المتعلقة بشبكات التدفق تجعل التدفق الآن معتمدًا كليًا على الوصلات. إن هذه المقاربة أكثر بداهة من التدفق الشبكي net flow المستخدم في الإصدارين السابقين.
- أصبح الفصل المتعلق بعمليات المصفوفات أصغر مما كان عليه في الإصدار الثاني نتيجة نقل المادة المتعلقة بأساسيات المصفوفات وخوارزمية Strassen إلى فصول أخرى.
- عدَّلنا طريقة معالجتنا لخوارزمية Knuth-Morris-Pratt لمطابقة المتتاليات الحرفية.
- صححنا عدة أخطاء، نشرت معظمها على موقع الوب الخاص بتصحيح أخطاء الإصدار الثاني، وبقي بعضها دون نشر.
- اعتمادًا على العديد من الطلبات، غيَّرنا التركيب النحوي لشبه الرماز عما كان عليه. نستخدم الآن "=" للتعبير عن الإسناد assignment، و "==" لاختبار المساواة، كما في C و C++ و Java و Python. وحذفنا، كذلك، الكلمات المفتاحية **do** و **then** واصطلحنا على "//" باعتباره رمزًا للتعليقات الممتدة حتى نهاية السطر. نستخدم الآن أيضًا التدوين النقطي dot-notation للدلالة على واصفات الغرض object attributes. يبقى شبه الرماز إجرائيًّا، وليس غرضي التوجه. وبعبارة أخرى، بدلاً من تنفيذ الطرائق على الأغراض، نستدعي الإجراءات ببساطة مع تمرير الأغراض باعتبارها موسطات.
- أضفنا 100 تمرين جديد و 28 مسألة جديدة. كذلك حدَّثنا العديد من المراجع وأضفنا عدة مراجع جديدة.
- في النهاية، راجعنا الكتاب بكامله، وأعدنا صياغة الجمل، والفقرات، والمقاطع لجعل الكتابة أكثر وضوحًا وفعاليةً.

موقع الوب:

يمكنك استخدام موقع الوب الخاص بالكتاب http://mitpress.mit.edu/algorithms/ للحصول على معلومات إضافية وللتواصل معنا. يتضمن موقع الوب وصلات إلى لائحة الأخطاء المعروفة، وإلى حلول لتمارين ومسائل مختارة، و(بالطبع) إلى لائحة تشرح نكات الأساتذة السخيفة، إضافة إلى محتويات أخرى قد نضيفها. يدلكم موقع الوب أيضًا على كيفية الإبلاغ عن الأخطاء أو التزويد بالمقترحات.

كيف أنتجنا هذا الكتاب:

أُنتج الإصدار الثالث، مثل الإصدار الثاني، باستخدام LATEX2ε. استخدمنا البنط Times مع مجموعة أنماط رياضية باستخدام البنوط MathTime Pro 2. نشكر Michael Spivak من شركة Publish or Perish، و Lance Carnes من شركة Personal Tex، و Tim Tregubov من كلية Dartmouth College للدعم التقني. جمعنا – كما في الإصدارين السالفين – دليل المصطلحات index باستخدام Windex، وهو برنامج كتبناه بلغة C، وجرى إنتاج المراجع باستخدام BIBTEX. وأنشأنا ملفات PDF لهذا الكتاب على حاسوب MacBook نظام تشغيله OS 10.5.

رسمنا الرسوم التوضيحية للإصدار الثالث باستخدام MacDraw Pro، حيث وضعت بعض التعابير الرياضية في الرسوم التوضيحية باستخدام حزمة psfrag المخصصة لـ LATEX2ε. لسوء الحظ، كانت MacDraw Pro برمجية قديمة، ولم تعد تسوَّق منذ عقد. غير أنه لحسن الحظ، كانت لدينا بعض حواسيب الماكنتوش التي تشتغل ضمن بيئة ماكنتوش كلاسيك بنظام تشغيل OS 10.4، وبذلك يمكنها تشغيل MacDraw Pro. وقد وجدنا أن استخدام MacDraw Pro ضمن بيئة كلاسيك، أسهل بكثير من أية برمجية رسم أخرى لأنماط الرسومات التي تصاحب عادة النصوص في علوم الحاسوب، وهي تنتج خرجًا جميلاً.[1] من يدري حتى متى ستستمر حواسيبنا ماكنتوش (من عهود ما قبل Intel) بالعمل، لذلك إذا كان هناك من يسمعنا من شركة Apple: *"رجاءً اصنعوا نسخة من MacDraw Pro تتوافق مع نظم التشغيل الأخرى."*

شكر خاص بالإصدار الثالث:

نحن نعمل مع مطبعة MIT Press منذ ما يزيد على عقدين حتى الآن، ويالها من علاقة ممتازة! نحن نشكر Ellen Faran، و Bob Prior، و Ada Brunstein، و Mary Reilly لمساعدتهم ودعمهم.

لقد كنا متباعدين جغرافيًا أثناء إنتاج الإصدار الثالث، حيث كنا نعمل في قسم علوم الحاسوب في كلية Dartmouth، وفي مخبر علوم الحاسوب والذكاء الصنعي في MIT، وقسم الهندسة الصناعية وبحوث العمليات في جامعة Columbia، ونحن نشكر جامعاتنا تلك وزملاءنا لتوفيرهم بيئات عمل محفزة وداعمة.

مرة أخرى، أنقذتنا Julie Sussman، من جمعية .P.P.A، بفضل جهودها في التصحيح قبل الطبع؛ فقد

[1] لقد جربنا عدة برامج رسم تشتغل ضمن بيئة Mac OS X، ولكن كان لكل منها نقائص مقارنةً ببرنامج MacDraw Pro. بإيجاز، حاولنا إنتاج رسوم هذا الكتاب التوضيحية باستخدام برنامج رسمٍ آخر مشهور جدًّا، ولكننا وجدنا بأنه استغرق خمسة أضعاف الوقت، على الأقل، مقارنةً ببرنامج MacDraw Pro لإنتاج كل رسم توضيحي، ولم تكن الرسوم بالجودة نفسها. بناء على ذلك، كان قرارنا بالتحول إلى تشغيل MacDraw Pro على حواسيب ماكنتوش قديمة.

ذهلنا مرارًا من الأخطاء التي فاتتنا وكشفتها Julie. وساعدتنا Julie أيضًا على تحسين عرضنا في أماكن عديدة، ولو كان هناك مسابقة لانتخاب مشاهير في مجال التحرير التقني، فلا ريب أنها ستكون أول من ينتخب. فهي مدهشة! شكرًا شكرًا شكرًا يا Julie! كذلك عثر Priya Natarajan على بعض الأخطاء التي استطعنا تصحيحها قبل إرسال الكتاب إلى المطبعة. فإن بقيت أية أخطاء (وحتمًا، لازال هناك البعض) فهي مسؤولية المؤلِّفين (ربما أُضيفت بعد أن قرأت Julie مادة الكتاب).

استُقيتْ معالجةُ أشجار van Emde Boas من مسودات Erik Demaine، التي تأثرت بدورها بـ Michael Bender. وقد أضفنا في هذا الإصدار أيضًا أفكارًا من Javed Aslam وBradley Kuszmaul و Hui Zha.

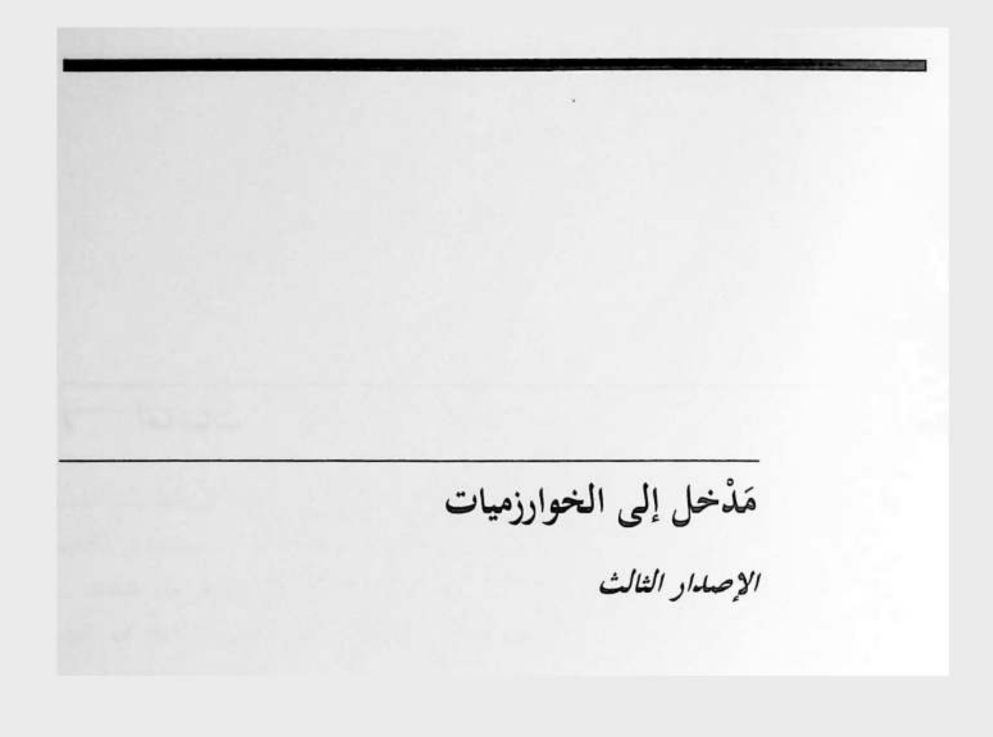
اعتمد الفصل المتعلق بتعدد النياسب على مسودات كُتبت أصلاً بالمشاركة مع Harald Prokop. تأثرت مادة الكتاب بعدة أعمال أخرى ضمن مشروع Cilk في MIT، ومنها أعمال Bradley Kuszmaul و Matteo Frigo. واستُوحي تصميمُ شبه الرماز المتعددُ النياسب من توسيعات Cilk الخاصة بمعهد MIT للغة C، ومن توسيعات ++Cilk الخاصة بشركة Cilk Arts للغة ++C.

نشكر أيضًا العديد من قرّاء الإصدارين الأول والثاني الذين بلّغوا عن أخطاء أو قدموا اقتراحات لتحسين هذا الكتاب. وقد صحّحنا جميع الأخطاء الفعلية التي جرى التبليغ عنها، وضمنّا ما استطعنا من الاقتراحات. ونحن سعداء بأن عدد هؤلاء المساهمين أصبح كبيرًا إلى درجة يتعذر تعداد أسمائهم جميعًا، وهذا ما نأسف بشأنه.

في النهاية، نشكر زوجاتنا: Nicole Cormen و Wendy Leiserson و Gail Rivest و Rebecca Ivry، وأولادنا: Ricky، و Will، و Debby، و Katie Leiserson؛ Alex، و Christopher Rivest؛ Molly، و Noah، و Benjamin Stein لحبهم ودعمهم لنا خلال مدة تحضيرنا لهذا الكتاب. لقد ساهم صبرهم وتشجيعهم في جعل هذا المشروع ممكنًا. نحن نهديهم هذا الكتاب مع حبنا.

Lebanon, New Hampshire	من	THOMAS H. CORMEN
Cambridge, Massachusetts	من	CHARLES E. LEISERSON
Cambridge, Massachusetts	من	RONALD L. RIVEST
New York, New York	من	CLIFFORD STEIN

شباط 2009



مَدْخل إلى الخوارزميات

الإصدار الثالث

I أساسيات

تمهيد

إن هذا الباب من الكتاب سيجعلك تبدأ بالتفكير في تصميم الخوارزميات وتحليلها؛ فقد أُعدَّ ليكون مقدمة سهلة في كيفية توصيف الخوارزميات، ومقدمة لبعض استراتيجيات التصميم التي سنستخدمها في الكتاب، وللكثير من الأفكار الأساسية المستخدمة في تحليل الخوارزميات. وستُبْنى الأجزاء التالية على هذه القاعدة.

يعطي الفصل الأول نظرة شاملة عن الخوارزميات وموقعها في النظم الحاسوبية الحديثة؛ فهو يُعَرِّف الخوارزميةَ ويسرد بعض الأمثلة. ويُبَيِّن كذلك أننا سنعتبر الخوارزميات تقانةً، تمامًا كالبنية المادية السريعة، وواجهات المستخدم البيانية، والنُّظم الغرضية التوجه، والشبكات.

سنصادف في الفصل الثاني خوارزمياتنا الأولى التي تَحلُّ مسألة فرز متتالية من n عددًا. هذه الخوارزميات مكتوبة بشبه رماز pseudocode، غير قابل للترجمة مباشرة إلى أية لغة برمجة مألوفة، إلا أنه ينقل بنية الخوارزمية بوضوح كاف يُمَكِّنُكَ من تنفيذها بلغة البرمجة التي تختارها. إن خوارزميات الفرز التي سندرسها هي خوارزمية الفرز بالإدراج insertion sort، التي تعتمد مبدأً تدريجيًّا incremental approach، وخوارزميةَ الفرز بالدمج merge sort التي تَستَخدم تقنيةً عَوْديَّة تُعرَف باسم "فَرِّقْ-تَسُدْ divide-and-conquer". وسنلاحظ من هذه الدراسة أنه على الرغم من ازدياد زمن التنفيذ اللازم لكلتا الخوارزميتين مع ازدياد حجم المسألة n، فإن معدل الزيادة يختلف في كل منهما. وسنحدِّد في هذا الفصل أزمنة التنفيذ running times هذه، وسَنُطوِّر طريقة مفيدة لتدوين هذه الأزمنة للتعبير عنها.

يعرِّف الفصلُ الثالث هذا التدوينَ تعريفًا دقيقًا، والذي نسميه التدوين المقارب asymptotic notation. يبدأ الفصل بتعريف عدة تدوينات مقاربة، نستخدمها لِحَدِّ أزمنة تنفيذ الخوارزمية من الأعلى و/أو من الأسفل. أما بقية الفصل الثالث، فهي في المقام الأول عرضٌ للتدوين الرياضي mathematical notation، الغرض منه التأكُّد أن استخدامك للتدوين يطابق طريقة التدوين المُعتَمَدَة في هذا الكتاب، وليس مجرد أن تتعلَّم أفكارًا رياضية جديدة.

يغوص الفصل الرابع بعمقٍ أكبر في طريقة "فَرِّق-تَسُدْ" التي تم التطرق إليها في الفصل الثاني. ويوفر أمثلة إضافية على خوارزمياتها، تشمل طريقة Strassen المدهشة لضرب مصفوفتين مربعتين. ويحتوي هذا الفصل أيضًا على طرائق لحل العلاقات العَوْديَّة، وهي مفيدة في وصف أزمنة تنفيذ الخوارزميات العَوديَّة. إحدى التقنيات الفعالة هي "الطريقة الرئيسة master method" التي سنستَخدمها غالبًا لحل العلاقات العَوديّة التي تَرد في خوارزميات فرِّق-تَسُد. ومع أن معظم الفصل الرابع مُخَصَّصٌ لإثبات صحة "الطريقة الرئيسة"، إلا أنه يُمكنُكَ الآن تجاوز هذا البرهان وأن تظل تستخدم "الطريقة الرئيسة".

يعرض الفصلُ الخامس التحليلَ الاحتمالي والخوارزميات ذات العشوائية المضافة. نَستخدم عادةً التحليل الاحتمالي لتحديد زمن تنفيذ خوارزميةٍ ما في الحالات التي قد يختلف فيها زمن تنفيذ الخوارزمية لمُدْخَلات مختلفة لها الحجم نفسه، وذلك بسبب وجود توزيع احتمالي أصيل في جوهر المسألة. في بعض الحالات نفترض أن قيم الدخل تتبع توزيعًا احتماليًّا معروفًا، وهذا ما يسمح لنا بحساب متوسط زمن التنفيذ على جميع المُدْخلات الممكنة. في حالات أخرى، لا يأتي التوزيع الاحتمالي من قيم الدخل بل من الخيارات العشوائية التي تُتَّخَذُ في سياق الخوارزمية. أما الخوارزميةُ التي لا يَتَحدد سلوكها من اختلاف قيم الدخل فقط، بل بالاعتماد على قيم يُولِّدُها مولِّد أعدادٍ عشوائية، فتسمى خوارزميةً ذات "عشوائية مضافة". نستطيع استخدام خوارزميات ذات عشوائية مضافة لجعل المُدْخلات تَتبع توزيعًا احتماليًّا – وهكذا نضمن عدم وجود دخل خاص يتسبب دائمًا بأداء سيِّئ – أو لتعيِّن حدود معدل خطأ الخوارزميات التي يسمح لها بإعطاء نتائج خاطئة بقدر محدود.

تحتوي الملاحق (أ)-(ت) مادة رياضية إضافية ستجدها ذات فائدة عندما تقرأ هذا الكتاب. ومن المرجح أنك عاينت جزءًا كبيرًا من محتوى فصول الملحق قبل قراءتك لهذا الكتاب (علمًا بأن التعريفات والمصطلحات التدوينية الخاصة التي نستخدمها قد تختلف أحيانًا عما رأيته سابقًا)، لذا ينبغي أن تتعامل مع الملاحق على أنها مادةٌ مرجعية تَعودُ إليها عند الحاجة. من ناحية أخرى، يُحتمل أنك لم تطَّلع من قبل على معظم محتوى الباب I؛ فجميع فصول هذا الباب والملاحق مكتوبة بأسلوب تعليمي مبسط.

1 دور الخوارزميات في الحوسبة

ما هي الخوارزميات؟ وما أهمية دراستها؟ وما دورها بالنسبة إلى بقية التقانات المستخدمة في الحواسيب؟ سنجيب في هذا الفصل عن هذه الأسئلة.

1.1 الخوارزميات

الخوارزمية algorithm عمومًا هي أي إجراء مُحَوْسب مُعَرَّف جيدًا، يأخذ قيمة أو مجموعة قيم نسميها ***الدخل input***، ويُنْتِجُ قيمة أو مجموعة قيم نسميها ***الخرج output***. فالخوارزمية إذًا هي متتالية محدودة من الخطوات المُحَوْسبة تُحوِّل الدخل إلى خرج.

يمكن أن ننظر إلى الخوارزمية على أنها أداة لحل ***مسألة مُحَوْسبة computational problem*** مُوَصَّفَة جيدًا. يُصِف نص المسألة العلاقة المطلوبة بين الدخل والخرج باستخدام مصطلحات عامة. تصف الخوارزمية إجراءً محوسبًا محددًا لتحقيق علاقة الدخل/الخرج تلك.

مثلاً قد نحتاج إلى فرز متتالية من الأعداد حسب ترتيبٍ غير متناقص. يتكرر ظهور هذه المسألة في الواقع العملي وتُوَفِر التربة الخصبة لعرض كثير من أدوات التحليل وتقانات التصميم القياسية. وفيما يلي عرض لكيفية تعريف ***مسألة الفرز sorting problem*** تعريفًا صوريًا:

الدخل: متتالية من n عددًا $\langle a_1, a_2, \ldots, a_n \rangle$.

الخرج: تبديل (إعادة ترتيب) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ متتالية الدخل بحيث يكون $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

مثال: ليكن لدينا متتالية الدخل $\langle 31, 41, 59, 26, 41, 58 \rangle$، تُنْتِجُ خوارزميةٌ ما للفرز في خرجها المتتالية $\langle 26, 31, 41, 41, 58, 59 \rangle$. تسمى متتالية الدخل المماثلة للمتتالية السابقة ***مُنْتَسَخ instance*** مسألة الفرز. يتكون ***مُنْتَسَخ مسألة ما instance of a problem*** عمومًا من الدخل اللازم لحساب حل هذه المسألة (ويُحقق أية شروط مفروضة في نص المسألة).

يُعدُّ الفرزُ عمليةً أساسية في علم الحاسوب، لأن برامج عديدة تَستَخدِمُها باعتبارها خطوة مرحلية.

ونتيجة لذلك، لدينا عدد كبير من خوارزميات الفرز الجيدة. ويعتمد اختيار الخوارزمية الفضلى لتطبيقٍ محدد على عدد من العوامل، منها: عدد العناصر التي ينبغي فرزها، ومدى ترتيب هذه العناصر سلفًا، والقيود الممكنة على قيم العناصر، ومعمارية الحاسوب، ونوع تجهيزة التخزين المستخدمة: ذاكرة رئيسة، أو أقراص، أو أشرطة مغناطيسية.

نقول عن خوارزمية ما إنها ***صحيحة correct*** إذا توقفت عند الخرج الصحيح لكل منتسخ دخل، ونقول إن الخوارزمية الصحيحة ***تَحُلُّ solves*** المسألة المُحَوْسبة المعطاة. أما الخوارزمية الخاطئة، فقد لا تتوقف على الإطلاق عند بعض منتسخات الدخل، أو قد تتوقف عند جواب غير صحيح. وعلى عكس ما قد تتوقعه، قد تكون بعض الخوارزميات الخاطئة مفيدة في بعض الحالات، إذا أَمْكَنَنا التَّحكُّمُ في معدل خطئها. سنرى مثالاً على خوارزمية يمكن التحكم في معدل خطئها في الفصل 31 عندما ندرس الخوارزميات المستخدمة لإيجاد أعداد أولية كبيرة. إلا أننا سوف نهتم في الحالة العامة بالخوارزميات الصحيحة فقط.

يمكن أن تُوَصَّفُ الخوارزمية باللغة الإنكليزية، كبرنامج حاسوبي، أو حتى كتصميم مادي. الشرط الوحيد أن يُقَدِّمَ التوصيفُ وصفًا دقيقًا للإجراء المُحَوْسَب الواجب اتباعه.

ما هي أنواع المسائل التي يمكن حلها باستخدام الخوارزميات؟

ليست مسألة الفرز هي المسألة المُحوسبة الوحيدة التي طُوِّرَت من أجلها الخوارزميات (قد يكون ذلك قد ورد في ذهنك عندما رأيت حجم هذا الكتاب)، إذ تشمل التطبيقات العملية للخوارزميات كل المجالات ومنها الأمثلة التالية:

- حقق مشروع الجينوم البشري Human Genome Project تقدمًا كبيرًا باتجاه أهداف تعريف كل الـ 100,000 جين الموجودة في الحمض النووي البشري DNA، وتحديد متتاليات 3 مليارات من الأزواج الكيميائية الأساسية التي تُكوِّن الحمض النووي البشري، وتخزين هذه المعلومات في قاعدة معطيات، وتطوير أدوات لتحليل هذه المعطيات. تتطلب كل من هذه الخطوات خوارزميات معقدة. ومع أن حلول المشاكل المختلفة التي تشملها هذه الخطوات تقع خارج نطاق هذا الكتاب، إلا أن كثيرًا من طرائق حل هذه المسائل الحيوية تستخدم أفكارًا من عدة فصول في هذا الكتاب، وهكذا تُتيح للعلماء تحقيق مهمات باستخدام الموارد المتاحة بفعالية. يؤدي استخدام خوارزميات فعالة إلى التوفير في وقت الإنسان وفي وقت الآلة، وفي المال عندما يمكن استنتاج معلومات أكثر من التقنيات المخبرية.
- تتيح الإنترنت للمستخدمين في جميع أنحاء العالم نفاذًا سريعًا واستعادة كميات كبيرة من المعلومات. وبمساعدة خوارزميات ذكية، أصبحت الصفحات على الإنترنيت قادرة على إدارة ومعالجة هذه الكميات الكبيرة من المعطيات. تشمل الأمثلة على المسائل التي تَستخدم الخوارزميات أساسًا لها مسألة إيجاد المسارات الجيدة التي يجب أن تُنْقَلَ عليها المعطيات (تَظْهَر تقانات حل مثل هذه المسائل في

الفصل 24)، ومسألة استخدام محرك بحث لإيجاد صفحاتٍ فيها معلوماتٌ معينة بسرعة (التقنيات المتعلقة بذلك موجودة في الفصل 11 والفصل 32).

- تتيح التجارة الإلكترونية التفاوض على البضائع والخدمات وتبادلها الكترونيًا، وتعتمد على خصوصية المعلومات الشخصية مثل أرقام البطاقات الائتمانية، وكلمات السر، والكشوف المصرفية. تشمل التقانات الأساسية المستخدمة في التجارة الإلكترونية تشفير المفتاح العام public-key cryptography والتواقيع الرقمية digital signatures (المشروحتان في الفصل 31)، اللّتانِ تعتمدان على الخوارزميات الرقمية numerical algorithms ونظرية الأعداد number theory.
- غالبًا ما تحتاج مشاريع التصنيع والمشاريع التجارية الكبيرة الأخرى إلى تخصيص الموارد النادرة بأكثر الطرق نفعًا. فقد ترغب شركة نفط في معرفة أماكن وضع الآبار ليكون ربحها المتوقع أكبر ما يمكن. وقد يرغب مرشح سياسي في أن يحدد أين ينفق الأموال في شراء الدعاية الانتخابية لزيادة فرص الكسب في الانتخابات قدر الإمكان. وقد ترغب شركة طيران في توزيع أطقمها على الرحلات بأقل الطرق كلفةً، بحيث تتأكد أنها تلبي حاجة كل رحلة طيران وأنها تراعي الأنظمة الحكومية المتعلقة بجدولة الطاقم. وقد يرغب مزود خدمة إنترنت في تحديد مكان وضع موارد إضافية لخدمة زبائنه بفاعلية أكبر. كل هذه الأمثلة هي مسائل يمكن أن تُحَلّ باستخدام البرمجة الخطية، التي سوف نَدْرُسُها في الفصل 29.

ومع أن بعض تفاصيل هذه الأمثلة تقع خارج نطاق هذا الكتاب، إلا أننا نَعْرض التقنيات الأساسية التي تنطبق على هذه المسائل وعلى مجالاتها. سنتطرق أيضًا في هذا الكتاب إلى كيفية حل كثير من المسائل الخاصة، منها:

- ليكن لدينا خارطة طريق حُدِّدَتْ عليها المسافة بين كل زوج من التقاطعات المتجاورة، ونرغب في تحديد أصغر طريق يصل بين تقاطعين. يمكن أن يكون عدد الطرق كبيرًا جدًّا، حتى لو استثنينا الطرق التي تتقاطع مع نفسها. كيف يمكن اختيار أقصر الطرق؟ هنا تُنَمْذَجُ خارطة الطريق (التي هي نفسها نموذج للطرق الحقيقية) كبيان graph (سنجده في الباب VI والملحق ب)، ونرغب في إيجاد أقصر مسار من عقدة إلى أخرى في البيان. سوف نرى كيف يمكن حل هذه المسألة بفعالية في الفصل 24.
- لتكن لدينا متتاليتان مرتبتان من الرموز، $X = \langle x_1, x_2, \dots, x_m \rangle$ و $Y = \langle y_1, y_2, \dots, y_n \rangle$ ونرغب في إيجاد أطول متتالية جزئية مشتركة من X و Y. إن المتتالية الجزئية من X ليست إلا X كاملة أو ربما مع حذف بعض عناصرها أو ربما حذفها كلها. على سبيل المثال، إحدى المتتاليات الجزئية من $\langle A, B, C, D, E, F, G \rangle$ هي $\langle B, C, E, G \rangle$. تعطي أطول متتالية جزئية مشتركة من X و Y قياسًا لمدى تشابه هاتين المتتاليتين. على سبيل المثال، لو كانت المتتاليتان زوجين أساسيين في جدائل DNA، عندها قد نعتبرهما متشابهتين إذا كان لهما متتالية جزئية مشتركة طويلة. لو احتوت X على m رمزًا و Y على n

رمزًا، حينها تحتوي X على 2^m و Y على 2^n متتالية جزئية ممكنة، على الترتيب. إن انتخاب جميع المتتاليات الجزئية الممكنة من X و Y ومطابقة بعضها مع بعضها الآخر يمكن أن يستغرق زمنًا طويلاً جدًّا لدرجة لا يمكن السماح به ما لم تكن m و n صغيرتان جدًّا. سنرى في الفصل 15 كيفية استخدام طريقة عامة لحل هذه المسألة بفعّالية أكبر بكثير تُعرَفُ بالبرمجة الديناميكية.

- ليكن لدينا تصميم ميكانيكي متعلق بمكتبة قطع، بحيث يمكن أن تحتوي كل قطعة منتسخات من قطع أخرى، ونرغب في سرد القطع وفق ترتيبٍ بحيث تظهر كل قطعة قبل أية قطعة تستخدمها. إذا كان التصميم مؤلفًا من n قطعة، عندها سيكون هنالك $n!$ من الترتيبات الممكنة، حيث تشير $n!$ إلى دالة العاملي. ولما كانت دالة العاملي تنمو بسرعة أكبر من الدالة الأسية، فإننا لا نستطيع أن نُوَلِّدَ عمليًّا كل ترتيب ممكن ثم نتحقق، ضمن ذلك الترتيب، أن كل قطعة تظهر قبل القطع التي تستخدمها (ما لم يكن لدينا سوى بِضْع قطعٍ فقط). هذه المسألة منتسخ من الفرز الطبولوجي، وسنرى في الفصل 22 كيفية حل هذه المسألة بفعالية.
- لتكن لدينا n نقطة في مستوٍ ونرغب في إيجاد غلاف محدب لهذه النقاط. الغلافُ المحدبُ هو أصغر مضلع محدب يحتوي النقاط. حدسيًّا، يمكن أن نتخيل كل نقطة ممثلة بمسمار مغروز على لوح. يتمثل عندها الغلاف المحدب بشريط مطاطي مشدود يحيط بجميع المسامير. كل مسمار يلتف حوله الشريط المطاطي هو رأس في الغلاف المحدب. (مثلاً انظر الشكل 33.6 في الجزء الثاني من الكتاب.) إن أيَّ مجموعةٍ من المجموعات الجزئية من النقاط وعددها 2^n يمكن أن تكوِّن رؤوس الغلاف المحدب. إن معرفة نقاط رؤوس المحدب لا يكفي، لأنه يلزمنا أيضًا معرفة ترتيب هذه النقاط. وهكذا توجد كثير من الخيارات لرؤوس الغلاف المحدب. يعطي الفصل 33 طريقتين جيدتين لإيجاد الغلاف المحدب.

إن الأمثلة السابقة غيض من فيض (كما قد تستدل من حجم هذا الكتاب) لكن هذه الأمثلة تُبْرِزُ صفتين شائعتين تشترك فيهما كثير من المسائل الخوارزمية المثيرة للاهتمام.

1. لهذه المسائل كثير من الحلول المرشحة، الغالبية العظمى منها لا تحل المسألة. وقد يمثل إيجاد الحل الذي يحل المسألة، أو الحل الذي يمكن اعتباره "الأفضل"، تحديًا كبيرًا.
2. لهذه المسائل تطبيقات عملية. أسهل الأمثلة على ذلك، من بين المسائل التي تعرضنا لها، تطبيقات مسائل أقصر المسارات. تهتم أية شركة شحن، سواء باستخدام الشاحنات أو السكك الحديدية ماديًّا بإيجاد أقصر المسارات عبر شبكة طرق أو شبكة سكك حديدية، لأن المسارات الأقصر تتطلب عملاً ووقودًا أقل. أو قد تحتاج عقدة تسيير على الإنترنت إلى إيجاد أقصر مسار عبر الشبكة لتوجيه رسالة ما بسرعة. أو أن شخصًا يرغب في قيادة سيارته من نيويورك إلى بوسطن قد يريد إيجاد اتجاهات القيادة من صفحة وِب مناسبة، أو أنه قد يستخدم نظام تحديد الموقع الجغرافي GPS الخاص به أثناء القيادة.

لا يوجد لكل مسألة قابلة للحل باستخدام الخوارزميات مجموعة مُعَرَّفة سهلة من الحلول المرشحة. على سبيل المثال، افترض أن لدينا مجموعة من القيم العددية تمثل عينات من إشارة، ونريد حساب تحويل فورييه المتقطع لهذه العينات. يقوم تحويل فورييه المتقطع بتحويل المجال الزمني إلى مجال ترددي، منتجًا مجموعة من المعاملات العددية، بحيث يمكننا تحديد قوة مختلف الترددات في الإشارة المأخوذة منها العينات. تمتلك تحويلات فورييه المتقطعة تطبيقات في ضغط المعطيات وجداء كثيرات الحدود والأعداد الصحيحة الضخمة، إضافة إلى كونها تمثل نواة معالجة الإشارة. يعرض الفصل 30 خوارزمية فعالة، هي تحويل فورييه السريع (يسمى شيوعًا FFT)، لهذه المسألة، ويعرض مخططًا لتصميم دارة الكيان الصلب لحساب تحويل فورييه السريع.

بنى المعطيات

يحتوي هذا الكتاب أيضًا العديد من بنى المعطيات. ***بنية المعطيات Data structure*** هي طريقة تخزين وتنظيم المعطيات لتسهيل الوصول إليها وإجراء تعديلات عليها. لا توجد بنية معطيات وحيدة تصلح لجميع الاحتياجات، لذلك يجب معرفة نقاط قوة وحدود استخدام الكثير من بنى المعطيات.

أسلوب عمل

على الرغم من أنك تستطيع استخدام هذا الكتاب ككتاب وصفات "cookbook" لخوارزميات جاهزة، فقد تصادف في يوم ما مسألة لا تستطيع أن تجد لها خوارزمية جاهزة منشورة (على سبيل المثال كثير من تمارين ومسائل هذا الكتاب). سوف يُعَلِّمُك هذا الكتاب تقنيات تحليل وتصميم الخوارزميات بحيث تستطيع تطوير خوارزميات بنفسك، وتبرهن أنها تعطي الجواب الصحيح، وتدرك مدى فعاليتها. تشرح الفصول المختلفة الأفكار المختلفة لحل المسائل الخوارزمية. وتشرح بعض الفصول مسائل خاصة، على سبيل المثال إيجاد الوسط الحسابي median وإحصائيات الترتيب في الفصل 9، وحساب شجرات المسح الصغرى في الفصل 23، وتحديد التدفق الأعظمي في شبكة في الفصل 26. تشرح الفصول الأخرى تقنيات، مثل فَرِّقْ-تَسُدْ في الفصل 4، والبرمجة الديناميكية في الفصل 15، والتحليل المخمَّد amortized analysis في الفصل 17.

مسائل صعبة

يعالج الجزء الأكبر من هذا الكتاب خوارزميات فعالة. مقياسنا الاعتيادي للفعالية هو السرعة، بمعنى آخر، الزمن الذي تستغرقه خوارزمية ما للوصول إلى النتيجة. مع ذلك هنالك بعض المسائل لا يُعْرَفُ لها حل فعّال. يدرس الفصل 34 مجموعة جزئية هامة من هذه المسائل، تعرف باسم NP-complete.

لماذا تعتبر مسائل NP-complete هامة؟ أولاً، ومع أنه لم توجد قطّ أية خوارزمية فعالة لمسألة من هذا النوع، لم يُثْبِتْ أحد عدم إمكانية وجود خوارزمية فعالة لهذه المسائل. وبعبارة أخرى، لا علم لنا بوجود

خوارزميات فعالة لمسائل NP-complete. ثانيًا، تمتلك مجموعة مسائل NP-complete خاصية لافتة للنظر، وهي أنه إذا وجدت خوارزمية فعالة لإحداها، عندها توجد خوارزميات فعالة لجميع مسائل هذه المجموعة. إن هذه العلاقة بين مسائل NP-complete جعلت التغلب على النقص في الحلول الفعالة جميعها أكثر إثارة للتحدي. ثالثًا، تشابه بعض مسائل NP-complete، المسائل التي نعرف لها حلولاً فعالة من مسائل الخوارزميات الفعالة إلا أنها مختلفة عنها. وقد أثار اهتمام علماء الحاسوب كيف أن تغييرًا بسيطًا في طرح المسألة قد يسبب تغييرًا كبيرًا في فعالية أفضل خوارزمية معروفة.

ينبغي أن تتعرّف مسائل NP-complete لأنه من المدهش أن بعضها يبرز كثيرًا في التطبيقات الواقعية. إذا ما طُلب إليك ذات يوم إيجاد خوارزمية فعالة لمسألة NP-complete، فعلى الأرجح أنك ستقضي وقتًا طويلاً في بحث غير مثمر. فإن استطعت إثبات أن المسألة هي NP-complete، فمن المجدي أن تصرف وقتك في تطوير خوارزمية فعالة تعطي حلاً جيدًا، ولو كان ليس أفضل حلٍّ ممكن.

وإليك مثالاً واقعيًا: لنفترض أن شركة توزيع بضائع لها مستودع مركزي. تملأ الشركة شاحناتها بالبضائع في المستودع المركزي وترسلها لتوزيع البضائع على عدة عناوين يوميًّا. يجب أن تُتِمَّ الشاحنةُ جولتها في نهاية اليوم وتعود إلى محطة الانطلاق لتكون جاهزة للتحميل في اليوم التالي. تريد الشركة – بهدف خفض الكلف – اختيار ترتيب نقاط توقُّف كلّ شاحنة للتوزيع بحيث تقطع أقصر مسافة ممكنة. هذه المسألة هي "مسألة البائع الجوال" المعروفة، وهي مسألة NP-complete، وليس لها خوارزمية فعالة معروفة. ومع ذلك، نعرف وفق فرضيات محددة خوارزميات فعالة تعطي مسافة إجمالية لا تزيد كثيرًا عن أصغر مسافة ممكنة. يناقش الفصل 35 "خوارزميات التقريب" هذه.

التوازي

استطعنا – لسنوات عديدة – أن نُدخل في حسابنا المعدلَ الثابت في ازدياد سرعات ساعة المعالج. غير أن الحدود الفيزيائية تمثل عائقًا أساسيًا في طريق التزايد المستمر في سرعات الساعة؛ وذلك بسبب تزايد كثافة الطاقة بمعدلٍ لاخطيٍّ يفوق تزايد سرعة الساعة، حيث تتعرض الرقاقات chips لخطر الانصهار بمجرد أن تصبح سرعات ساعتها كبيرة كفاية. ولتنجيز حسابات أكثر بالثانية، لا تُصَمَّمُ الرقاقات لتحتوي نواة معالجة واحدة فقط بل عدة نوى معالجة. ونستطيع أن نشبه هذه الحواسيب المتعددة النوى multicore بعدة حواسيب تسلسلية على رقاقة مفردة؛ وبكلمات أخرى، هي نوع من الحواسيب المتوازية parallel computers. وللحصول على التنجيز الأفضل للحواسيب المتعددة النوى، نحتاج عادةً إلى تصميم خوارزميات ونحن نضع فكرة التوازي في أذهاننا. يعرض الفصل 27 نموذجًا للخوارزميات "المتعددة النياسب multithreaded"، التي تستفيد من النوى المتعددة. لهذا النموذج فوائد من وجهة نظر نظرية، ويشكل القاعدة لعدة برامج حاسوب ناجحة، تشمل برنامجًا لبطولة الشطرنج.

تمارين

1-1.1

أعط مثالاً واقعيًّا يحتاج إلى الفرز، أو مثالاً واقعيًّا يحتاج إلى حساب الغلاف المحدب convex hull.

2-1.1

ما هي قياسات الفعالية الأخرى غير السرعة التي قد تُستَخدم في الواقع؟

3-1.1

اختر بنية معطيات عاينتها سابقًا، وناقش نقاط قوتها وحدود استخدامها.

4-1.1

كيف تكون مسألتا أقصر مسار والبائع الجوال الواردتان آنفًا متشابهتين؟ وكيف تكونان مختلفتين؟

5-1.1

ابحث في مسألة حقيقية لا بد من البحث عن حلها الأفضل، ثم ابحث في مسألة يكون أفضل حلٍّ تقريبي لها حلاً جيدًا إلى حدٍّ بعيد.

2.1 الخوارزمياتُ بصفتها تقانةً

إذا افترضنا أن سرعة الحواسيب لانهائية وأن ذاكرة الحواسيب مجانية، فهل ثمة سببٌ لدراسة الخوارزميات؟ الجواب نعم، لا لسبب إلا لأنك ترغب في إثبات أن طريقة الحل التي تقترحها تتوقف، وأنها تتوقف بإعطاء النتيجة الصحيحة.

إذا كانت سرعة الحواسيب لانهائية، فإن أية طريقة صحيحة لحل مسألة ما سوف تؤدي الغرض. لكن قد ترغب في أن يكون تنجيزُك للحل يطبق عمليًّا هندسة البرمجيات (على سبيل المثال يجب أن يكون تنجيزك مصممًا وموثقًا جيدًا)، لكنك ستستخدم على الأغلب أسهل الطرق لتنجيزها.

من البديهي أن الحواسيب قد تكون فائقة السرعة، لكنها ليست ذات سرعة لانهائية. وقد تكون ذاكرة الحاسوب غير غالية الثمن، لكنها ليست مجانية. لذلك فإن زمن الحساب هو مورد محدود، وكذلك حجم الذاكرة. ولهذا السبب ينبغي استخدام هذه الموارد بحكمة، وستساعدك الخوارزميات الفعالة في الزمن والذاكرة على تحقيق هذا الغرض.

الفعالية

كثيرًا ما تختلف الخوارزميات المخصصة لحل المشكلة نفسها في فعاليتها efficiency. يمكن أن تكون هذه الاختلافات ملموسة أكثر من الاختلافات الناتجة عن البنى المادية hardware والبرمجيات software.

كمثال على ذلك، سنرى في الفصل الثاني خوارزميتيْ فرز. تُعرَفُ الخوارزمية الأولى ***بالفرز بالإدراج insertion sort***، ويستغرق تنفيذها زمنًا يعطى تقريبًا بالعلاقة c_1n^2، وذلك لترتيب n عنصرًا، حيث c_1 ثابت لا يعتمد على n. أي إنها تستغرق زمنًا يتناسب مع n^2. أما الخوارزمية الثانية، فهي ***الفرز بالدمج merge sort***، ويستغرق تنفيذها زمنًا يعطى تقريبًا بالعلاقة $c_2n\lg n$، حيث $\lg n$ تعني $\log_2 n$، و c_2 ثابت آخر لا يعتمد أيضًا على n. يكون ثابت الفرز بالإدراج عادةً أصغر من ثابت الفرز بالدمج، أي $c_1 < c_2$. وسنرى لاحقًا أن للعوامل الثابتة تأثيرًا على زمن التنفيذ أقل بكثير من تأثير حجم المسألة n. فإذا كتبنا زمن تنفيذ الفرز بالإدراج بالعلاقة $c_1n\cdot n$، وزمن تنفيذ الفرز بالدمج بالعلاقة $c_2n\cdot\lg n$، عندها نجد أنه فيما يحتوي الفرز بالإدراج العامل n في زمن تنفيذه، يحتوي الفرز بالإدراج العامل $\lg n$ وهو أصغر بكثير من سابقه. (فمثلاً، إذا كان $n = 1000$، فإن $\lg n$ يساوي 10 تقريبًا، وإذا كان $n = 10^6$، فإن $\lg n$ لا يتجاوز 20.) إن الفرز بالإدراج ينفِّذ عادة أسرع من الفرز بالدمج في حالات حجوم الدخل الصغيرة، ولكن ما إن يصبح حجم المسألة n كبيرًا كفاية، حتى تصبح فائدة العامل $\lg n$ في الفرز بالدمج مقارنةً بـ n أكثر من تعويض الفرق في العوامل الثابتة c_1 و c_2؛ فمهما كانت قيمة c_1 أصغر من c_2، فتوجد دومًا نقطة تقاطع يكون بعدها الفرز بالدمج أسرع.

لنأخذ مثالاً واقعيًا نقارن فيه بين حاسوب سريع نسبيًا (حاسوب A) ينفِّذ خوارزميةَ الفرز بالإدراج، وبين حاسوب بطيء نسبيًا (حاسوب B) ينفِّذ خوارزميةَ الفرز بالدمج. يُطلب إلى كلتا الخوارزميتين فرز صفيفة مكونة من عشرة ملايين عدد. (قد يبدو أن عشرة ملايين عدد هائل، لكنْ إذا كانت الأعداد صحيحة وممثلة باستخدام 8 بايت لكل منها، فسيَشغل الدخل نحوًا من 80 ميغا بايت، وهذا يمكن تخزينه حتى في ذاكرة الحاسوب المحمول الرخيص الذي يتسع لأضعاف هذا الحجم.) لنفترض أن الحاسوب A ينفذ 10 مليارات تعليمة في الثانية (أسرع من أي حاسوب تسلسلي بمفرده في زمن تأليف هذا الكتاب)، والحاسوب B ينفذ فقط عشرة ملايين (10^7) تعليمة في الثانية، أي إن الحاسوب A أسرع من الحاسوب B بألف مرة من حيث القدرات الحاسوبية الأولية. ولجعل الفرق أكثر وضوحًا، لنفترض أن أمهر مبرمج في العالم بَرْمَجَ خوارزمية الفرز بالإدراج بلغة الآلة الخاصة بالحاسوب A، وأن الرماز الناتج يتطلب $2n^2$ تعليمة لفرز n عددًا (هنا $c_1 = 2$)، ولنفترض إضافةً إلى ذلك أنه بَرْمَجَ خوارزميةَ الفرز بالدمج مبرمجٌ متوسط المهارة بلغة برمجةٍ عليا تَستخدم مترجمًا غير فعّال بحيث يُنتج رمازًا يتطلب $50n\lg n$ تعليمة. عندها نجد أنه لترتيب 10 ملايين عدد يستغرق الحاسوب A:

$$\frac{2.(10^7)^2 \text{ تعليمة}}{10^{10} \text{ تعليمة/ثانية}} = 20{,}000 \text{ ثانية (أكثر من 5.5 ساعة)}\ ,$$

ويستغرق الحاسوب B:

$$\frac{50 . 10^7 \lg 10^7 \text{ تعليمة}}{10^7 \text{ تعليمة/ثانية}} \approx 1163 \text{ ثانية (أقل من 20 دقيقة)}$$

وهكذا نجد أنه باستخدام خوارزمية ذات زمن تنفيذ بطيء، ومترجم ضعيف، فإن الحاسوب B ينفِّذ هذه الخوارزمية أسرع بـ 17 مرة من الحاسوب A! على أن فائدة الفرز بالدمج تَظهر بوضوح أكبر في حال فرز 100 مليون عدد: ففي حين يستغرق الفرز بالإدراج أكثر من 23 يومًا، يستغرق الفرز بالدمج أقل من أربع ساعات. وبوجه عام، تزداد الفائدة النسبية للفرز بالدمج بازدياد حجم المسألة.

الخوارزميات والتقانات الأخرى

يُظهر المثال السابق أنه ينبغي أن نتعامل مع الخوارزميات على أنها ***تقانة technology*** شأنها في ذلك شأن الكيان المادي للحاسوب. يعتمد الأداء العام للنظام على اختيار خوارزميات فعالة بقدر اعتماده على اختيار الكيان المادي السريع للحاسوب. وقد حدث تقدم سريع في الخوارزميات تمامًا كما حدث في تقانات الحاسوب الأخرى.

قد تتساءل: هل الخوارزميات هامة بالفعل في الحواسيب المعاصرة مع وجود التقانات المتقدمة الأخرى مثل:

- بنى الحاسوب المتقدمة وتقانات التصنيع.
- واجهات المستخدم البيانية المألوفة وسهلة الاستخدام GUI.
- النظم الغرضية التوجه Object Oriented Systems.
- تقانات الويب المكاملة Integrated Web Technologies.
- التشبيك السريع، السلكي واللاسلكي Fast Networking, both Wired and Wireless.

الجواب نعم. لأنه على الرغم من أن وجود بعض التطبيقات التي لا تتطلب صراحة محتوًى خوارزميًّا في مستوى التطبيق (مثل بعض التطبيقات البسيطة المعتمدة على الوب)، فإن كثيرًا منها يتطلب ذلك. لنأخذ على سبيل المثال خدمة معتمدة على الوب تحدِّد كيفية السفر من مكان إلى آخر. إن تحقيق هذه الخدمة قد يقوم على كيان مادي سريع، وواجهة مستخدم بيانية، وتشبيك واسع، وكذلك على إمكانية التوجه بالأغراض. إلا أنه يتطلب أيضًا خوارزميات لبعض العمليات، مثل إيجاد الطرق (ربما استخدام خوارزمية حساب أقصر مسار)، وإظهار خرائط الترجمة، واستقراء العناوين.

أضف إلى ذلك أن التطبيق، ولو كان لا يتطلب محتوًى خوارزميًّا في مستوى التطبيق، فإنه يَعتمد بقوة

على الخوارزميات. فالتطبيقُ يَعتمد على كيانٍ ماديٍّ سريع، وتصميمُ الكيان المادي يَستخدم بدوره الخوارزميات. والتطبيق يعتمد على واجهات مستخدمٍ بيانية، وتصميمُ أية واجهةٍ بيانية يعتمد بدوره على الخوارزميات. والتطبيق يعتمد على الشبكات، والتسييرُ routing يعتمد بدوره بكثافة على الخوارزميات. وأخيرًا، التطبيق يُكتَب بلغة رماز الآلة machine code، أي يعالجه مترجمٌ compiler أو مفسِّرٌ interpreter أو مُجمِّع assembler، وجميعُها تعتمد بدورها بكثافة على الخوارزميات. وهكذا فإن الخوارزميات موجودة في صلب معظم التقانات المستخدمة في الحواسيب المعاصرة.

يضاف إلى ذلك أنه مع الازدياد المستمر في قدرات الحواسيب، فإننا نستخدمها لحل مسائل أكبر لم يسبق لنا أن حللناها. وكما رأينا في المقارنة بين الفرز بالإدراج والفرز بالدمج، تصبح الفروق في الفعالية بين الخوارزميات محسوسةً بوضوح مع أحجام المسائل الكبيرة.

إن امتلاك قاعدة متينة من المعرفة والتقنية الخوارزمية تمثل عاملَ فضْلٍ لتمييز المبرمجين المهرة حقًّا من المبتدئين؛ فباستخدام تقانة حوسبة حديثة يمكنك إنجاز بعض المهمات دون معرفة الكثير عن الخوارزميات، ولكنك تستطيع إنجاز الكثير الكثير إذا امتلكت خلفيةً خوارزمية جيدة.

تمارين

1-2.1

أعط مثالاً عن تطبيق يتطلب محتوًى خوارزميًّا في مستوى التطبيق، وناقش دور الخوارزميات المعْنية.

2-2.1

افترض أننا نقارن بين تنجيز الفرز بالإدراج والفرز بالدمج على الآلة نفسها. يتم الفرز بالإدراج بـ $8n^2$ خطوة لمدخلات حجمها n، على حين يتم الفرز بالدمج بـ $64n\lg n$ خطوة. ما هي قيم n التي يتفوق فيها الفرز بالإدراج على الفرز بالدمج؟

3-2.1

ما هي أصغر قيمة لـ n بحيث يكون تنفيذُ خوارزميةٍ زمنُ تنفيذها $100n^2$ أسرعَ من خوارزميةٍ زمنُ تنفيذها 2^n على الحاسوب نفسه؟

مسائل

1-1 مقارنة زمن التنفيذ

حدِّد – لكل دالةٍ $f(n)$ وزمنٍ t في الجدول التالي – أكبر قيمة لـ n لمسألةٍ يمكن حلها خلال الزمن t، بافتراض أن الخوارزمية المستخدمة لحل المسألة تستغرق $f(n)$ ميكروثانية.

	1 ثانية	1 دقيقة	1 ساعة	1 يوم	1 شهر	1 سنة	1 قرن
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

ملاحظات الفصل

هنالك الكثير من مصادر المعلومات الممتازة عن موضوع الخوارزميات العام، منها: Aho و Hopcroft و Ullman [5, 6]؛ Baase و Van Gelder [28]؛ Dasgupta و Brassard و Bratley [55]؛ Dasgupta و Papdimitriou و Vazirani [83]؛ Goodrich و Tamassia [148]؛ Hofri [178]؛ Horowitz و Sahni و Rajasekaran [181]؛ Johnsonbaugh و Schaefer [193]؛ Kingston [205]؛ Kleinberg و Tardos [208]؛ Knuth [209, 210, 211]؛ Kozen [220]؛ Levitin [235]؛ Manber [242]؛ Mehlhorn [249, 250, 251]؛ Purdom و Brown [287]؛ Reingold و Nievergelt و Deo [293]؛ Sedgewick [306]؛ Sedgewick و Flajolet [307]؛ Skiena [318]؛ و Wilf [356]. ويناقش كلٌّ من Bently [42, 43] و Gonnet [145] بعضَ أكثر الجوانب عمليةً في تصميم الخوارزميات. توجد أيضًا دراسات تمسح حقل الخوارزميات في كتيب علم الحاسوب النظري *Handbook of Theoretical Computer Science, Volume A* [342]، وكتيب CRC في الخوارزميات ونظرية الحوسبة Algorithms and Theory of Computation Handbook [25]. وتوجد أيضًا لمحة عن الخوارزميات المستخدمة في علم الأحياء المحوسب في كتب Gusfield [156] و Pevzner [275] و Setubal and Meidanis [310] و Waterman [350].

2 لنبدأ

سيطلعك هذا الفصل على المنهجية التي سنستخدمها في هذا الكتاب للتفكير في تصميم وتحليل الخوارزميات. هذا الفصل مستقل بذاته، لكنه يحتوي عدة إشارات لمقاطع ستعرض في الفصلين الثالث والرابع. (يحتوي كذلك عدة مجاميع سلمية، يبين الملحق (أ) كيفية حلها.)

سنبدأ بتفحص خوارزمية الفرز بالإدراج insertion sort لحل مسألة الفرز المعروضة في الفصل الأول. نعرّف "شبه الرماز" pseudocode الذي ينبغي أن يكون معروفًا للقراء الذين عملوا في برمجة الحاسوب، ونستخدمه لإيضاح كيف سنوصف خوارزمياتنا. بعد توصيف الخوارزمية، نبرهن صحة فرزها، ونحلل زمن تنفيذها. يقدم التحليل تدوينًا notation يسلط الضوء على كيفية ازدياد زمن التنفيذ مع زيادة عدد الحدود التي يجب فرزها. بعد مناقشة الفرز بالإدراج، نعرض مفهوم فرِّقْ تَسُد لتصميم الخوارزميات ونستخدمه لتطوير خوارزمية تسمى الفرز بالدمج merge sort. ثم ننهي الفصل بتحليل زمن تنفيذ خوارزمية الفرز بالدمج.

1.2 الفرز بالإدراج

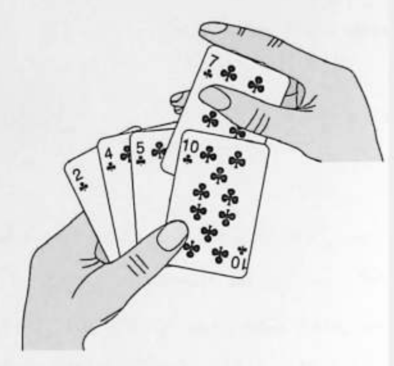
تحل خوارزميتنا الأولى، الفرز بالإدراج، ***مسألة الفرز sorting problem*** الواردة في الفصل الأول:

الدخل: متتالية من n عددًا $\langle a_1, a_2, \ldots, a_n \rangle$.

الخرج: تبديل (إعادة ترتيب) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ متتالية الدخل بحيث يكون $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

تُعرَفُ الأعداد التي نرغب في ترتيبها أيضًا ***بالمفاتيح Keys***. ومع أننا من حيث المبدأ نرتب متتالية، غير أن الدخل الذي يرد إلينا يكون على شكل صفيفة من n عنصرًا.

سنصف في هذا الكتاب الخوارزميات عمومًا كبرامج مكتوبة ***بشِبه رماز pseudocode*** يشابه في عدة جوانب C، أو C++، أو Java، أو Python، أو Pascal. فإذا كنت قد اطلعت على أيٍّ من هذه اللغات، فلن تجد عناءً كبيرًا في قراءة خوارزمياتنا. إن ما يفرّق شبه الرماز عن الرماز "الحقيقي"، هو أننا نستخدم في شبه الرماز الطريقة المعبرة الأكثر وضوحًا وإيجازاً لتوصيف الخوارزمية المعطاة. قد تكون اللغة الإنكليزية هي



الشكل 1.2 ترتيب أوراق اللعب باستخدام الفرز بالإدراج

أوضح طريقة، لذا لا تندهش إذا صادفتَ عبارةً أو جملةً إنكليزية مضمنة في مقطع من الرماز "الحقيقي". ثمة فرقٌ آخر بين شبه الرماز والرماز الحقيقي، هو أن شبه الرماز لا يُعنى عادةً بقضايا هندسة البرمجيات. وللتعبير عن جوهر الخوارزمية بإيجاز أكبر، غالباً ما نتجاهل قضايا تجريد المعطيات، والنسيقية **modularity**، ومعالجة الخطأ.

سنبدأ ***بالفرز بالإدراج*** ***insertion sort***، الذي هو خوارزمية فعالة لفرز عدد صغير من العناصر. يعمل الفرز بالإدراج وفق الطريقة التي يفرز بها كثير من لاعبي الورق أوراق اللعب. نبدأ بيد يسرى فارغة وأوراق اللعب وجهها إلى الأسفل باتجاه الطاولة. ثم نرفع في كل مرة ورقة واحدة من الطاولة وندرجها (نحشرها) في الموضع الصحيح في اليد اليسرى. لإيجاد الموضع الصحيح للورقة، نقارنها بكل ورقة في اليد، من اليمين إلى اليسار، كما هو موضح في الشكل 1.2. وتكون الأوراق الموجودة في اليد اليسرى مفروزة في كل المرات، وكانت هذه الأوراق أصلاً الأوراق العلوية للكومة الموجودة على الطاولة.

سنعرض شبه رمازنا لخوارزمية الفرز بالإدراج على هيئة إجراء اسمه INSERTION-SORT، يأخذ موسطًا هو صفيفة $A[1..n]$ تحتوي على متتالية طولها n يُطلب ترتيبها. (يشار في الرماز إلى عدد العناصر n في A بـ $A.length$.) تَفرز الخوارزميةُ أعدادَ الدخل ***في المكان*** ***in place***: وتعيد ترتيب الأعداد ضمن الصفيفة A، مع تخزين عدد ثابت منها على الأكثر خارج الصفيفة في أي وقت. عندما ينتهي الإجراء INSERTION-SORT سوف تحتوي صفيفةُ الدخل A متتاليةَ الخرج المفروزة.

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
```

```
4        i = j − 1
5        while i > 0 and A [i] > key
6            A[i + 1] = A [i]
7            i = i − 1
8        A[i + 1] = key
```

لامتغيرات الحلقة وصحة الفرز بالإدراج

يوضح الشكل 2.2 كيف تعمل هذه الخوارزمية في حالة الصفيفة $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. يشير الدليل j إلى "الورقة الحالية" التي تدرج في اليد. في بداية كل تكرار من تكرارات حلقة **for** التي متحول تحكمها هو j، تكوّن الصفيفة الجزئية المؤلَّفة من العناصر $A[1\,..\,j-1]$ أوراق اليد المفروزة حاليًّا، وتطابق الصفيفة الجزئية المتبقية $A[j+1\,..\,n]$ كومة الأوراق التي ماتزال على الطاولة. إن العناصر $A[1\,..\,j-1]$ هي في الواقع نفس العناصر التي في المواضع 1 حتى $j-1$ *أصلاً*، لكنها الآن بترتيب مفروز. نصوغ هذه الخواص للصفيفة الجزئية $A[1\,..\,j-1]$ صوريًّا بصفتها ***لامتغير حلقة loop invariant***:

> في بداية كل تكرار من تكرارات حلقة **for** التي تشمل الأسطر 1-8، تتألف الصفيفة الجزئية $A[1\,..\,j-1]$ من العناصر التي هي أصلاً في هذه الصفيفة الجزئية، لكن بترتيب مفروز.

نستخدم لامتغيرات الحلقة لتساعدنا على فهم سبب كون خوارزمية ما صحيحة. وهنا يجب أن نوضح ثلاثة أمور عن لامتغير الحلقة:

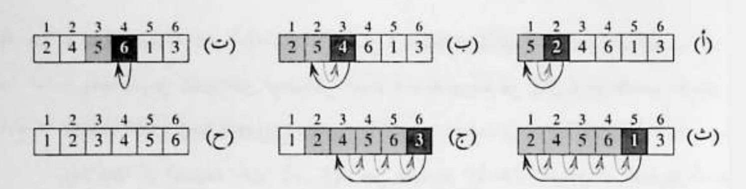
الاستبداء: يكون لامتغير الحلقة صحيحًا قبل التكرار الأول للحلقة.

المحافظة: إذا كان لامتغير الحلقة صحيحًا قبل تكرارٍ ما للحلقة، فيبقى صحيحًا قبل التكرار التالي.

الإنهاء: عندما تنتهي الحلقة، يعطي لامتغير الحلقة خاصية مفيدة تساعد على بيان أن الخوارزمية صحيحة.

عند تحقُّق الخاصيتين الأولى والثانية، يكون لامتغير الحلقة صحيحًا قبل كل تكرار من تكرارات الحلقة. (طبعًا، لدينا الحرية في استخدام خصائص محققة أخرى غير لامتغير الحلقة نفسه لإثبات أن لامتغير الحلقة يبقى صحيحًا قبل كل تكرار.) لاحظ الشبه مع الاستقراء الرياضي، وهو أنه كي تُثْبِتَ تحقُّق خاصيةٍ ما، ينبغي أن تُثْبِتَ حالةً أساسية وخطوةً استقرائية. وهنا يَظهر أن اللامتغير يتحقَّق قبل أن يقابِل التكرارُ الأولُ الحالةَ الأساسية، ويَظهر أيضًا أن اللامتغير يتحقَّق من تكرارٍ إلى تكرارٍ يقابِل الخطوةَ الاستقرائية.

الخاصية الثالثة ربما تكون أهم خاصية، لأننا نستخدم لامتغير الحلقة لبيان الصحة. نستخدم عادة لامتغير الحلقة بالتلازم مع الشرط المسبب لإنهاء الحلقة. تختلف خاصية الإنهاء عن الكيفية التي نستخدم فيها عادة الاستقراء الرياضي، حيث نطبق الخطوة الاستقرائية حتى اللانهاية؛ أما هنا، فنوقف "الاستقراء" عندما تنتهي الحلقة.



الشكل 2.2 عمل الفرز بالإدراج على الصفيفة $A = \langle 5,2,4,6,1,3 \rangle$. تَظهَر دلائل العناصر فوق المستطيلات، وتَظهَر القيم المخزنة في مواضع الصفيفة داخل المستطيلات. تمثل الأشكال من (أ) إلى (ح) تكرارات حلقة **for** التي تشمل الأسطر 1-8. يحتفظ المستطيل الأسود في كل تكرار بالمفتاح (key) المأخوذ من $A[j]$، الذي يقارَن بالقيم الموجودة في المستطيلات المظلَّلة إلى يساره في اختبار السطر 5. تُظهر الأسهمُ المظللة قيمَ الصفيفة المزاحة موضعًا واحدًا إلى اليمين في للسطر 6، وتشير الأسهم السوداء إلى أين يتحرك المفتاح في السطر 8. يمثل الشكل (ح) الصفيفة النهائية المفروزة.

لنفحص كيفية تحقُّق هذه الخواص في حالة الفرز بالإدراج.

الاستبداء: نبدأ ببيان أن لامتغير الحلقة يتحقَّق قبل بدء التكرار الأول للحلقة، عندما يكون $j = 2$.[1] لذلك تتألف الصفيفة الجزئية $A[1..j-1]$ من العنصر $A[1]$ فقط، الذي هو في الواقع العنصر الأصلي في $A[1]$. إضافة إلى ذلك، فإن هذه الصفيفة الجزئية مفروزة (بداهة، بالطبع)، وهذا يبيِّن أن لامتغير الحلقة يتحقَّق قبل بدء التكرار الأول للحلقة.

المحافظة: بعد ذلك، نتناول الخاصية الثانية: أي بيان أن كل تكرار يحافظ على لامتغير الحلقة. بعبارة بسيطة، يقوم جسم الحلقة **for** على إزاحة $A[j-1]$، و $A[j-2]$، و $A[j-3]$ موقعًا واحدًا نحو اليمين المرة تلو الأخرى حتى يَجِدَ الموقعَ المناسبَ للعنصر $A[j]$ (الأسطر 4-7)، وعندها يُدْرِجُ قيمة $A[j]$ في هذا الموقع (السطر 8). تتألف عندها الصفيفة الجزئية $A[1..j]$ من العناصر الموجودة أصلاً في $A[1..j]$، لكن بترتيب مفروز. ويحافظ تزايد j في حالة التكرار التالي للحلقة for على لامتغير الحلقة.

تتطلب منا معالجة أكثر صورية للخاصية الثانية تحديد وبيان لامتغير حلقة لحلقة **while** الواقعة في الأسطر 5-7. عند هذه النقطة نفضل عدم الغوص حتى هذه الدرجة من الصوريّة، ونكتفي بتحليلنا المبسّط لبيان أن الحلقة الخارجية تحقق الخاصية الثانية.

الإنهاء: أخيرًا لنتفحص ما يَحصُل عندما تنتهي الحلقة. إن الشرط الذي يسبب انتهاء حلقة **for** هو

---

[1] عندما تكون الحلقة هي حلقة **for**، فإن اللحظة التي نختبر عندها لامتغير الحلقة – وهي بالضبط اللحظة التي تسبق أول تكرار – هي مباشرة بعد الإسناد الابتدائي لمتحول عداد الحلقة، وبالضبط قبل أول اختبار في ترويسة الحلقة. في حالة الفرز بالإدراج، يكون هذا الزمن بعد إسناد 2 للمتحول j لكن قبل أول اختبار فيما إذا كان $j \le A.length$.

$j > A.length = n$. ولما كان كلُّ تكرارٍ في الحلقة يزيد قيمةَ j بمقدار 1، فيجب أن نحصل على $j = n + 1$ في ذلك الحين. وبتعويض القيمة $n + 1$ عوضًا عن j في عبارة لامتغير الحلقة، نحصل على الصفيفة الجزئية $A[1..n]$ وهي تتألف من العناصر الموجودة في $A[1..n]$ أصلاً، ولكن بترتيب مفروز. وبملاحظة أن الصفيفة الجزئية $A[1..n]$ هي الصفيفة الداخلية، نستنتج أن الصفيفة الداخلية مفروزة. فالخوارزمية إذن صحيحة.

سنستخدم طريقة لامتغيرات الحلقة هذه لإظهار الصحة في هذا الفصل، وفي الفصول القادمة أيضًا.

اصطلاحات شبه الرماز

سوف نستخدم الاصطلاحات التالية في شبه رمازنا.

- تشير الإزاحة العمودية أثناء الكتابة إلى بنية كتلة. فمثلاً، يتألف جسم حلقة **for** الذي يبدأ عند السطر 1 من الأسطر 2-8، أما جسم حلقة **while** التي تبدأ عند السطر 5، فيحتوي السطرين 6 و 7، ولكن لا يحتوي السطر 8. ينطبق نموذجنا في الإزاحة العمودية أثناء الكتابة على تعليمات **if-else**[2] أيضًا. إن استخدام الإزاحة العمودية عوضًا عن المؤشرات الاصطلاحية لبنية الكتلة، مثل تعليمتَي **begin** و **end**، يقلِّل كثيرًا من مراكمة المصطلحات الإضافية على حين أنه يحافظ على الوضوح ، أو حتى يزيد منه.[3]
- تمتلك التراكيب الحلقية **while**، و **for**، و **repeat-until**، وتركيب الشرط **if-else** معاني مشابهة لمعانيها في لغات البرمجة C، و ++C، وجافا، و python، وباسكال.[4] في هذا الكتاب، يحتفظ عداد الحلقة بقيمته بعد الخروج من الحلقة، على عكس بعض الحالات التي تظهر في ++C وجافا وباسكال. وهكذا، تكون قيمة عداد الحلقة، مباشرة بعد حلقة **for**، هي أول قيمة تتجاوز حد حلقة **for**. لقد استخدمنا هذه الخاصية في برهاننا صحة الفرز بالإدراج. إن ترويسة حلقة **for** في السطر 1 هي **for** $j = 2$ **to** $A.length$، وهذا يعني أنه عندما تنتهي هذه الحلقة، يكون $j = A.length + 1$ (أو،

---

[2] في تعليمة **if-else**، نزيح **else** عموديًا إلى نفس مستوى **if** المطابقة لها. وعلى الرغم من أننا نحذف الكلمة المفتاحية **then**، فإننا نشير أحيانًا إلى الجزء المنفذ – عندما تكون نتيجة الاختبار الذي يتبع **if** صحيحة – على أنه عبارة **then**. في حالة الاختبارات المتعددة الفروع، نستخدم **elseif** لنشير إلى الاختبارات الواردة بعد الاختبار الأول.

[3] يظهر كل إجراء شبه رمازي في هذا الكتاب على صفحة واحدة بحيث لا تضطر للتمييز بين مستويات الإزاحة العمودية في رماز مُفرق على صفحات.

[4] تمتلك معظم اللغات المهيكلة ككتل block-structured languages تراكيب مكافئة، لذا قد تختلف القواعد الدقيقة لصياغتها. تفتقد لغة python حلقات **repeat-until**، وتختلف حلقات **for** قليلاً في طريقة عملها عن حلقات **for** في هذا الكتاب.

ما يكافئه $j = n + 1$ ، لأن $n = A.length$). نستخدم الكلمة المفتاحية **to** عندما تزيد حلقةُ **for** عدادَ حلقتها في كل تكرار، ونستخدم الكلمة المفتاحية **downto** عندما تُنقص حلقةُ **for** عدادَ حلقتها. وعندما يتغير عداد الحلقة بمقدار أكبر من 1، يَتْبَعُ مقدارُ التغيرِ الكلمةَ المفتاحية الاختيارية **by**.

- يشير الرمز "//" إلى أن ما تبقّى من السطر هو تعليق.
- الإسناد المتعدد من الشكل $i = j = e$ يسند لكلٍّ من المتحولين i و j قيمة العبارة e؛ ويجب أن تعالج كإسناد $j = e$ متبوعٍ بالإسناد $i = j$.
- المتحولات (مثل i و j و key) هي متحولات محلية بالنسبة إلى إجراء معطى. ولن نستخدم متحولات عامة (global variables) دون الإشارة إليها صراحة.
- يجري الوصول إلى عناصر الصفيفة بتحديد اسم الصفيفة متبوعًا بالدليل ضمن قوسين مربعين. مثال، $A[i]$ يشير إلى العنصر ذي الترتيب i من الصفيفة A. الكتابة بالشكل ".." تستخدم للتعبير عن مجال من القيم ضمن صفيفة. ويُستخدم التدوين ".." للإشارة إلى مجال من القيم ضمن صفيفة؛ وعلى ذلك تشير $A[1..j]$ إلى صفيفة جزئية من A تتألف من j عنصرًا هي: $A[1], A[2], \dots, A[j]$.
- ننظم المعطيات المركبة عادةً ضمن ***أغراض Objects***، تتكون بدورها من ***واصفات attributes***. نَنْفُذُ إلى واصف محدد باستخدام التركيب النحوي الموجود في كثير من لغات البرمجة الغرضية التوجه: اسم الغرض، متبوعًا بنقطة، متبوعًا باسم الواصف. كمثال على ذلك، نعالج الصفيفة كغرض مع صفة *الطول* $length$ للإشارة إلى عدد العناصر التي تحتويها. لتحديد عدد العناصر في صفيفة A نكتب $A.length$.
- ونُعالِجُ المتحولَ الذي يمثل صفيفة أو غرضًا كمؤشر للمعطيات الممثلة للصفيفة أو الغرض. إن الإسناد $y = x$، يؤدي إلى جعل $y.f$ تساوي $x.f$ لكل الواصفات f لغرض ما x. إضافة إلى ذلك، إذا جعلنا الآن $x.f = 3$، بعدئذٍ لن يكون $x.f = 3$ فقط، بل $y.f = 3$ أيضًا. وبعبارة أخرى، تشير كل من x و y إلى الغرض نفسه بعد الإسناد $y = x$.
- يمكن أن يتتالى تدويننا للواصفات "cascade". فعلى سبيل المثال افترض أن الواصف f نفسه هو مؤشر لنوع ما من الأغراض g. عندها يحتوي التدوين $x.f.g$ ضمنيًّا أقواسًا من الشكل $(x.f).g$. وبعبارة أخرى، إذا أسندنا $y = x.f$، عندها تكون $x.f.g$ هي $y.g$ نفسها.
- قد لا يشير المؤشر أحيانًا إلى أي غرض على الإطلاق. في هذه الحالة، نعطيه القيمة الخاصة NIL.
- تُمَرَّرُ الموسطات إلى الإجراء ***بالقيمة by value***: يتلقى الإجراء المُستَدعَى نسخته الخاصة من الموسطات، وإذا أَسند قيمةً لموسط، فلن يَرى الإجراءُ المُسْتَدْعِي التغييرَ الحاصل. وعندما تُمَرَّرُ أغراض، يُنْسَخُ المؤشر للمعطيات المُمَثلة للغرض، لكن لا تنسخ واصفات الغرض. فعلى سبيل المثال، إذا كان x موسطًا

لإجراءٍ مُسْتَدْعًى، فلن يكون الإسناد $x = y$ داخل الإجراء المُسْتَدْعى مرئيًا للإجراء المُسْتَدْعِي. أما الإسناد $x.f = 3$، فهو مرئي. وبالمثل، تُمَرَّرُ الصفائف بالمؤشر، بحيث يُمَرَّر مؤشرٌ إلى صفيفة، بدلاً من الصفيفة كلها، وتكون تغيرات عناصر الصفيفة الفردية مرئية للإجراء المستدعِي.

- تعيد تعليمة **return** التحكم إلى نقطة الاستدعاء في الإجراء المُسْتدعِي. تأخذ أيضًا معظم تعليمات **return** قيمةً لتعيدها إلى المستدعي. يختلف شبه رمازنا عن كثير من لغات البرمجة في أننا نسمح بأن تعاد قيم متعددة في تعليمة **return** مفردة.
- تتصرف المعاملات البوليانية "and" و"or" مثل ***دارات قصر short circuiting***. وهذا يعني أنه عندما نقيِّم العبارة "x and y" فإننا نقيِّم x أولاً. فإذا كانت قيمة x هي FALSE، فلا يمكن عندها تقييم كامل العبارة بـ TRUE، لذا لا نقيِّم y. من ناحية أخرى، إذا كانت قيمة x هي TRUE، فعلينا تقييم y لتحديد قيمة العبارة كاملة. وبالمثل، نقيِّم y في العبارة "x or y" إذا كانت قيمة x هي FALSE فقط. تسمح لنا عوامل دارات القصر بكتابة عبارات بوليانية مثل: "$x \neq$ NIL and $x.f = y$" دون القلق مما يمكن أن يحدث عندما نحاول تقييم $x.f$ عندما تكون x هي NIL.
- تشير الكلمة المفتاحية **error** إلى أن خطأً ما قد حدث لأن شروط استدعاء الإجراء كانت خاطئة. ويكون الإجراء المستدعي هو المسؤول عن معالجة الخطأ، ومن ثم فإننا لا نحدد الفعل الذي يجب أن يتخذ.

تمارين

1-1.2

باستخدام نموذج الشكل 2.2، وضِّح عمل الفرز بالإدراج على الصفيفة $A = \langle 31, 41, 59, 26, 41, 58 \rangle$.

2-1.2

أعد كتابة إجراء الفرز بالإدراج للفرز بترتيب متناقص بدل الفرز بترتيب متزايد.

3-1.2

لتكن لدينا مسألة البحث searching problem التالية:

الدخل: متتالية من n عددًا $A = \langle a_1, a_2, a_3, \dots, a_n \rangle$ وقيمة ما v.

الخرج: دليل ما i بحيث $v = A[i]$، أو القيمة الخاصة NIL في حال عدم ظهور v في A.

اكتب شبه رماز يمثل ***البحث الخطي linear search***، الذي يمسح المتتالية A، بحثًا عن v. برهن باستخدام لامتغير الحلقة أن خوارزميتك صحيحة. تأكد أن لامتغير حلقتك يحقق الخصائص الثلاث الضرورية.

4-1.2
لتكن لدينا مسألة جمع عددين صحيحين اثنانيين كل منهما مكون من n بتًّا، مُخزَّنَيْن في صفيفتين A و B، تتألف كلٌّ منهما من n عنصرًا. يجب أن يُخزَّن مجموع العددين الصحيحين بصيغةٍ اثنانية في صفيفة C عدد عناصرها $(n + 1)$. صُغ المسألة صوريًّا، واكتب شبه رماز لجمع العددين الصحيحين.

2.2 تحليل الخوارزميات

أصبح ***تحليل analyzing*** الخوارزمية يعني التنبؤ بالموارد التي تحتاج إليها الخوارزمية عند تنفيذها. في بعض الأحيان، تشكل الموارد مثل الذاكرة، أو عرض حزمة الاتصال، أو البنى المادية للحاسوب الاهتمام الرئيس، لكن في معظم الأحيان يكون زمن الحساب هو الذي نريد قياسه. يمكننا عمومًا – عن طريق تحليل عدة خوارزميات مرشحة لحل مسألة ما – أن نُحدِّدَ بسهولةٍ أيّ الخوارزميات أكثر كفاءةً. وقد يُظهر هذا التحليل أكثرَ من حلٍّ قابل للتطبيق، لكنْ يمكننا أيضًا في كثير من الأحيان أن نستبعد عدة خوارزميات أقل كفاءةً.

قبل أن نستطيع تحليل خوارزمية ما، يجب أن نمتلك نموذجًا لتقانة التنجيز implementation التي سنستخدمها، ومن ذلك نموذج موارد هذه التقانة وكلفها. سنفترض في معظم هذا الكتاب، معالجًا وحيدًا عموميًّا، ونموذج ***آلة وصول عشوائي random-access machine (RAM)*** للحَوْسَبة باعتبارها تقانة تَنْجيز، مدركين أن خوارزمياتنا ستُنَجَّزُ باعتبارها برامج حاسوبية. تنفذ التعليمات في نموذج RAM تعليمة بعد تعليمة، دون وجود عمليات متسايرة concurrent operations.

إذا أردنا الحديث بدقة مطلقة، فعلينا تعريف تعليمات نموذج RAM وكلفها بدقة. إلا أن ذلك قد يكون مرهقًا، ولن يقدم إلا القليل من الرؤية داخل تحليل الخوارزمية وتصميمها. لذا يجب أن نكون متنبهين لعدم إساءة استعمال نموذج RAM. مثلاً، ماذا لو احتوت RAM تعليمة تقوم بالفرز؟ بمقدورنا عندئذ أن ننفذ عملية الفرز بتعليمة واحدة فقط. لن تكون مثل هذه الآلة واقعية، لأن الحواسيب الحقيقية لا تمتلك مثل هذه التعليمات. مرشدنا إذن في اختيار النموذج هو تصميم الحواسيب الحقيقية. يحتوي نموذج RAM تعليمات توجد عادة في الحواسيب الحقيقية: تعليمات حسابية (مثل: add و subtract و multiply و divide و remainder و floor و ceiling)، وتعليمات نقل معطيات (load و store و copy)، وتعليمات تحكم (التفريع الشرطي وغير الشرطي conditional and unconditional branch، واستدعاء الإجراء الفرعي subroutine call، والعودة return). تستغرق كلٌّ تعليمة منها مقدارًا ثابتًا من الزمن.

إن أنواع المعطيات في نموذج RAM هي الأعداد الصحيحة integer وذات الفاصلة العائمة floating point (لتخزين الأعداد الحقيقية). ومع أننا في العادة لا نشغل أنفسنا بدقة تعريف الأعداد في هذا الكتاب، غير أن الدقة تكون في بعض التطبيقات حاسمة. نفترض أيضًا وجود حدٍّ لحجم كل كلمة من المعطيات؛

فمثلاً، عند العمل على مدخلات من الحجم n، نفترض عادة أن الأعداد الصحيحة تُمثَّل بالمقدار $c \lg n$ بتًّا حيث $c \geq 1$ ثابت. نحتاج إلى $c \geq 1$ حتى تستطيع كلُّ كلمةٍ احتواءَ قيمة n، متيحة لنا التدليل على عناصر الدخل المستقلة، ونشترط أن يكون c ثابتًا بحيث لا ينمو حجم الكلمة كيفيًّا. (لو أمكن لحجم الكلمة أن ينمو كيفيًّا، لاستطعنا تخزين كميات ضخمة من المعطيات في كلمة واحدة ولأمكننا أن نعمل عليها جميعها في زمن ثابت. ومن الواضح أن هذا السيناريو غير واقعي.)

تحتوي الحواسيب الحقيقية تعليمات لم تُدرَج آنفًا، ومثل هذه التعليمات تُمثِّل المنطقةَ الرمادية في نموذج RAM. على سبيل المثال، هل الرفعُ إلى قوةٍ تعليمةٌ تَستغرق زمنًا ثابتًا؟ في الحالة العامة، لا؛ فهي تأخذ عدة تعليمات لحساب x^y عندما تكون x و y أعدادًا حقيقية. من جهة ثانية، يكون الرفعُ إلى قوة، في بعض الحالات المحددة، عمليةً تَستغرق زمنًا ثابتًا. تمتلك كثير من الحواسيب تعليمة الإزاحة إلى اليسار "shift left"، التي تزيح في زمن ثابت بتاتِ عددٍ صحيح k موضعًا إلى اليسار. في معظم الحواسيب، تكافئ عمليةُ إزاحةِ بتات عددٍ صحيحٍ موضعًا واحدًا إلى اليسار الضربَ بـ 2، أي إن إزاحة البتات k موضعًا إلى اليسار تكافئ الضربَ بـ 2^k. وهكذا، تستطيع مثل هذه الحواسيب حساب 2^k بتعليمة زمن ثابت واحدة بإزاحة العدد الصحيح 1 بـ k موضعًا إلى اليسار، مادامت k ليست أكثر من عدد البتات في كلمة الحاسوب. سنحاول تجنب مثل هذه المناطق الرمادية في نموذج RAM، لكن سنعالج حساب 2^k على أنها عملية زمن ثابت عندما يكون k عددًا صحيحًا موجبًا وصغيرًا كفاية.

لن نحاول في نموذج RAM نمذجةَ تراتبيةِ الذاكرة memory hierarchy الشائعة في الحواسيب الحالية. ذلك أننا لا ننمذج الذواكر الخابية caches memory أو الذاكرة الافتراضية virtual memory. تحاول بعض النماذج الحسابية أن تأخذ بالحسبان تأثيرات تراتبية الذاكرة، الهامَّة أحيانًا في البرامج الحقيقية على آلاتٍ حقيقية. جزءٌ ضئيل فقط من مسائل هذا الكتاب يعالج آثارَ تراتبيةِ الذاكرة، فيما لن يُعنى بها الجزء الأكبر من التحليلات في هذا الكتاب. إن النماذج التي تحتوي تراتبية الذاكرة أعقد فعلاً من نموذج RAM، ومن ثم يمكن أن يكون التعامل معها صعبًا. إضافة إلى أن تحليلات نموذج RAM هي عادة مُتَنبِّئاتٌ predictors ممتازة عن الأداء على الآلات الفعلية.

يمكن أن يشكل تحليل الخوارزمية – ولو كانت بسيطة – في نموذج RAM تَحَدِّيًا؛ فقد تشمل الأدواتُ الرياضية اللازمة: التحليلَ التوافيقي combinatorics، ونظريةَ الاحتمالات، وبراعةً في الجبر، والقدرة على تحديد أهم المصطلحات في العبارة. ولما كان من الممكن أن يكون سلوك خوارزميةٍ ما مختلفًا تبعًا لكل دخل ممكن، فإننا نحتاج إلى وسيلة لتلخيص هذا السلوك في عبارات بسيطة سهلة الفهم.

ومع أننا نختار عادةً نموذج آلة واحدة فقط لتحليل خوارزمية ما، فسنظل نواجه خيارات كثيرة في تحديد كيفية التعبير عن تحليلنا. لذا فإننا نرغب في طريقة بسيطة للكتابة والمعالجة تُظْهِرُ المميزات الهامة لمتطلبات خوارزمية ما من الموارد، وتُجَنِّبنا التفاصيل المملة.

تحليل خوارزمية الفرز بالإدراج

يعتمد زمن تنفيذ إجراء الفرز بالإدراج على حجم الدخل: إذ إنَّ فرْزَ ألف عددٍ يَستغرق زمنًا أطول من الزمن المستغرق لفرز ثلاثة أعداد. إضافةً إلى ذلك، قد يستغرق الفرز بالإدراج أزمنةً مختلفة لفرز متتاليتي دخل لهما الحجمُ نفسُه اعتمادًا على مدى قرب وضعهما الابتدائي من الفرز المطلوب. يزداد زمن تنفيذ الخوارزمية عمومًا مع ازدياد حجم الدخل، لذلك يوصف زمن تنفيذ برنامجٍ ما عادةً كدالةٍ لحجم دخله. ولفعل ذلك، نحتاج لأن نُعَرِّفَ بعناية مصطلحَي زمن التنفيذ وحجم الدخل.

يعتمد أفضل تعبير عن ***حجم الدخل* input size** على المسألة المدروسة. ففي أغلب المسائل، مثل: الفرز، أو حساب تحويلات فورية المتقطعة، يكون القياس الطبيعي الأكثر هو *عدد الحدود في الدخل*. مثال ذلك، حجم الصفيفة n في الفرز. وفي مسائل أخرى كثيرة، مثل ضرب عددين صحيحين، يكون القياس الأفضل لحجم الدخل هو *العدد الكلي للبتات* اللازم لتمثيل الدخل في التدوين الاثناني النظامي ordinary binary notation. وفي بعض الأحيان، يكون من الأنسب وصف حجم الدخل بعددين بدلاً من عدد واحد. فمثلاً، إذا كان دخل الخوارزمية بيانًا graph، يمكن أن يوصف الدخل بعدد عقد vertices البيان ووصلاته edges. سنبيِّن بوضوح مقياس حجم الدخل المستخدم في كل مسألة ندرسها.

أما ***زمن تنفيذ* running time** خوارزميةٍ ما على حجمِ دخلٍ محدد، فهو عدد العمليات الأولية أو "الخطوات steps" المنفَّذة. من المناسب هنا تعريف فكرة الخطوة بحيث تكون مستقلةً عن الآلة قدر الإمكان. سنعتمد، مؤقتًا، الفكرةَ التالية: يحتاج كلُّ سطرٍ من شبه رمازنا إلى مقدار ثابت من الزمن لتنفيذه. قد يستغرق سطرٌ ما قدرًا من الزمن مختلفًا عما يستغرقه سطرٌ آخر، لكننا سنفترض أن كلَّ تنفيذٍ للسطر ذي الرقم i يَستغرق زمنًا c_i، حيث c_i ثابت. تنسجم هذه الفكرة مع نموذج RAM، وتُظهِر أيضًا كيف يمكن أن يُنَجَّز شبه الرماز على معظم الحواسيب الفعلية.[5]

سوف يتطور في المناقشة التالية تعبيرنا عن زمن تنفيذ الفرز بالإدراج INSERTION-SORT من صيغة شائكة تَستخدم جميع تكاليف التعليمات c_i إلى تدوينٍ بسيط يكون أكثر إيجازًا وأسهل في المعالجة. وهذا التدوين البسيط سيجعل من السهل كذلك تحديد فعالية خوارزميةٍ ما بالنسبة إلى خوارزميةٍ أخرى.

سنبدأ بعرض إجراء الفرز بالإدراج مع "الكلفة" الزمنية لكل عبارة وعدد مرات تنفيذ كل عبارة. في حال

---

[5] توجد هنا بعض النقاط الدقيقة؛ فالخطوات المحوسبة التي نحددها بالإنكليزية هي غالبًا أشكالٌ متنوعة من إجراءٍ يتطلب أكثرَ من مجرد مقدارٍ ثابت من الزمن. فمثلاً، قد نستعمل في هذا الكتاب عبارة "افرز النقاط حسب الإحداثيات x"، وهذا، كما سنرى، يستغرق أكثر من مقدار ثابت من الزمن. وكذلك، فإن التعليمة التي تستدعي مساقًا فرعيًا تستغرق زمنًا ثابتًا، مع أن المساق الفرعي، بمجرد استدعائه، قد يستغرق زمنًا أطول. وهذا يعني أننا نَفْصل عملية ***استدعاء*** المساق الفرعي – أي تمرير الوسطاء إليه، إلخ – عن عملية ***تنفيذ*** المساق الفرعي.

حيث $j = 2, 3, \dots, n$ ،$n = A.length$ نرمز بـ t_j لعدد مرات تنفيذ حلقة الاختبار **while** لقيمة j في السطر الخامس. عند الخروج من حلقة **while** أو حلقة **for** بالطريقة العادية (بسبب الاختبار في ترويسة الحلقة)، ينفَّذ الاختبار مرةً واحدةً زيادةً على عدد مرات تنفيذ جسم الحلقة. نفترض أن التعليقات ليست عبارات تنفيذية، لذلك لا تأخذ زمن تنفيذ.

INSERTION-SORT(A)		*cost*	*times*
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1 \dots j-1]$.	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$

إن زمن تنفيذ الخوارزمية هو مجموعُ أزمنةِ تنفيذ كلِّ عبارة منفَّذة؛ فالعبارة التي تأخذ c_i خطوة لتنفيذها وتُنَفَّذ n مرةً ستساهم بمقدار $c_i n$ في زمن التنفيذ الكلي.[6] لحساب $T(n)$، وهو زمن تنفيذ الفرز بالإدراج، لدخلٍ مؤلَّف من n قيمة، نجمع جداءَ عمودَي *الكلفة والأزمنة*، فينتج:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

إذا كانت المدخلات من حجم معطى، فإن زمن تنفيذ الخوارزمية قد يعتمد على الدخل المُعْطى من ذلك الحجم. ففي INSERTION-SORT مثلاً، تحدث الحالة الفضلى إذا كانت الصفيفة مفروزة سابقًا. حيث نجد عندها أن $A[i] \leq key$ لكل $j = 2, 3, \dots, n$ في السطر الخامس عندما يأخذ المتحول i قيمته الابتدائية على أنها $j - 1$. وبهذا، تكون $t_j = 1$ لكل $j = 2, 3, \dots, n$ ، ويكون زمن التنفيذ في أفضل الحالات:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

---

[6] لا تصح هذه الخاصية بالضرورة لموردٍ كالذاكرة. فالعبارة التي تتعامل مع m كلمة من الذاكرة وتنفَّذ n مرة لا تستهلك بالضرورة mn كلمة ذاكرة إجمالاً.

نستطيع أن نعبر عن زمن التنفيذ هذا بالعلاقة $an + b$ حيث a و b ثابتان يتعلقان بكلف العبارات c_i؛ وهي بذلك ***دالة خطية* Linear function** في n.

أما إذا كانت الصفيفة مفروزة بترتيب عكسي – أي بترتيب متناقص – فنتنج عندها أسوأ الحالات. وعلينا عندها مقارنة كل عنصر $A[j]$ بكل عنصر من داخل الصفيفة الجزئية المفروزة $A[1..j-1]$، وهكذا يكون $t_j = j$ لكل $j = 2, 3, \dots, n$. وبملاحظة أن:

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

و

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

(انظر الملحق أ لمراجعة كيفية حل هذه المجاميع)، نجد أن زمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad -(c_2 + c_4 + c_5 + c_8) \ . \end{aligned}$$

ونستطيع أن نعبِّر عن زمن تنفيذ أسوأ الحالات بالعبارة $an^2 + bn + c$ حيث a و b و c ثوابت تتعلق أيضًا بكلف العبارات c_i، وبهذا فهي ***دالة من الدرجة الثانية* quadratic function** في n.

وكما في الفرز بالإدراج، يكون زمن تنفيذ خوارزمية ما عادةً ثابتًا لدخلٍ معطى، على الرغم من أننا سنجد في الفصول اللاحقة بعض الخوارزميات "ذات العشوائية المضافة" الممتعة التي يمكن أن يتغير سلوكها حتى في دخلٍ ثابت.

تحليل أسوأ الحالات والحالة الوسطى

تناولنا في تحليلنا للفرز بالإدراج كلاً من أفضل الحالات، التي تكون فيها صفيفة الدخل مفروزةً سلفًا، وأسوأ الحالات، التي تكون فيها صفيفة الدخل مفروزةً عكسيًّا. لكننا سنركز في بقية هذا الكتاب على إيجاد زمن تنفيذ أسوأ الحالات فقط، وذلك لأنه أطول زمن تنفيذٍ *لأيِّ* دخلٍ حجمه n. ونعطي هنا ثلاثة أسباب لهذا التوجُّه.

- يعطينا زمن تنفيذ أسوأ الحالات لخوارزميةٍ ما حدًّا أعلى لزمن التنفيذ لأيِّ دخل. وتعطينا معرفتُه ضمانة

أن الخوارزمية لن تستغرق زمنًا أطول منه. حيث لا نحتاج إلى تخمين زمن التنفيذ آملين ألاّ يصبح أسوأ من ذلك التخمين.

- في بعض الخوارزميات، يتكرر ورود أسوأ الحالات مرارًا؛ فمثلاً، أثناء البحث في قاعدة معطيات عن جزء محدد من المعلومات، ستَحدث أسوأ الحالات لخوارزمية البحث كثيراً عندما لا تكون المعلومات موجودة في قاعدة المعطيات. وفي بعض التطبيقات، قد يكون البحث عن معلومات غير موجودة كثير الحدوث.
- كثيرًا ما تكون "الحالة الوسطى" مماثلةً – إلى حدٍّ ما – لأسوأ الحالات في السوء. لنفترض أننا اخترنا عشوائيًا n عددًا، وأننا طبقنا الفرز بالإدراج، فكم ستستغرق الخوارزمية لتحديد موضع إدراج عنصر ما $A[j]$ في صفيفة جزئية $A[1..j-1]$؟ وسطيًّا، نصف العناصر في $A[1..j-1]$ هي أقل من $A[j]$، ونصف العناصر أكبر منه. لذا فإننا سنختبر في المتوسط نصف عناصر الصفيفة الجزئية $A[1..j-1]$، وبذلك يساوي t_j تقريبًا $j/2$. وسيؤول زمن تنفيذ الحالة الوسطى الناتج إلى دالةٍ من الدرجة الثانية في حجم الدخل، تمامًا كما هو الحال في زمن تنفيذ أسوأ الحالات.

سوف نهتم – في بعض الحالات الخاصة – بزمن تنفيذ ***الحالة الوسطى average-case*** للخوارزمية؛ وسنتعرّف تقانةَ التحليل الاحتمالي مطبقةً على عدة خوارزميات في هذا الكتاب. من ناحية ثانية، إن مجال تحليل الحالة الوسطى محدود، لأنه قد لا يكون واضحاً ما هو الدخل الذي يمثل دخل "الحالة الوسطى" لمسألة محددة. سنفترض، في الغالب، أن احتمال حدوث جميع المدخلات من حجم معين متساوٍ. وعمليّاً، قد لا تكون هذه الفرضية محققة، لكن يمكننا في بعض الأحيان استخدام ***خوارزمية ذات عشوائية مضافة randomized algorithm***، تقوم بخيارات عشوائية، لإتاحة التحليل الاحتمالي وحساب زمن تنفيذٍ ***متوقع expected***. سندرس الخوارزميات ذات العشوائية المضافة بتفصيل أكبر في الفصل الخامس وفي عدة فصول تالية أخرى.

مرتبة النمو

لقد استخدمنا بعض التجريدات المبسّطة لتسهيل تحليلنا لإجراء الفرز بالإدراج. من ذلك أننا تجاهلنا الكلفةَ الحقيقية لكل عبارة، وذلك باستخدام الثوابت c_i لتمثيل هذه الكلف. ثم لاحظنا أنه حتى هذه الثوابت تعطينا تفاصيل أكثر مما نحتاج إليه حقيقة: وعبَّرنا عن زمن تنفيذ أسوأ الحالات بالصيغة $an^2 + bn + c$ لبعض الثوابت a و b و c التي تعتمد على كلف العبارات c_i. و بهذا لم نتجاهل فقط الكلف الحقيقية للعبارة، بل الكلف المجردة c_i أيضًا.

سنقوم الآن بتجريد مبسَّطٍ إضافي: وهو ***معدل نمو rate of growth*** (أو ***مرتبة نمو order of growth***) زمن التنفيذ الذي يهمنا حقيقة. ولهذا السبب، نُهتم فقط بالحد الرئيسي leading term في الصيغة

(مثلاً an^2)، لأن الحدود ذات الدرجة الدنيا تكون غير مؤثرة نسبيًّا عند قيم n الكبيرة. كذلك، نهمل المعامل الثابت للحد الرئيسي، لأن العوامل الثابتة أقل تأثيرًا من معدل النمو في تحديد الفعالية الحسابية للمدخلات الكبيرة. ففي حالة الفرز بالإدراج، عندما نتجاهل الحدود ذات المرتبة الدنيا والمعامل الثابت الذي يسبق الحد الرئيسي، فإننا نُبقِي على العامل من الدرجة n^2 من الحد الرئيسي. ونقول إن للفرز بالإدراج زمن تنفيذ في أسوأ الحالات $\Theta(n^2)$ (تلفظ "ثيتا n مربع"). سنستخدم تدوين-Θ في هذا الفصل دون تعريف دقيق له، وسَنُعَرِّفُه بدقة في الفصل 3 .

نعتبر عادة أن خوارزمية ما أكثر فعالية من خوارزمية أخرى إذا كان لزمن تنفيذها في أسوأ الحالات مرتبة نمو أدنى. لكن، قد تستغرق خوارزميةٌ – لزمنِ تنفيذها مرتبةُ نموٍّ أعلى – زمناً أقل في حالة مدخلات أصغر من خوارزميةٍ لزمن تنفيذها مرتبةُ نموٍّ أدنى، وذلك بسبب العوامل الثابتة والحدود الأدنى مرتبة. إلا أنه، في حالة مدخلات كبيرة كفاية، ستنفَّذ الخوارزمية ذات الـ $\Theta(n^2)$ مثلاً، بسرعة أكبر في أسوأ الحالات من خوارزمية من المرتبة $\Theta(n^3)$.

تمارين

1-2.2

عبِّر عن الدالة $n^3/1000 - 100n^2 - 100n + 3$ باستخدام التدوين-Θ .

2-2.2

لتكن لدينا مسألة فرز n عددًا مخزنًا في صفيفة A، وذلك بأن نوجد أولاً أصغر عنصر في A، ونبادله مع العنصر $A[1]$، ثم نوجد ثاني أصغر عنصر في A، ونبادله مع العنصر $A[2]$. ونستمر بهذه الطريقة للعناصر الـ $n-1$ الأولى من A. اكتب شبه رماز لهذه الخوارزمية، التي تُعْرَفُ ***بالفرز الانتقائي selection sort***. ما هو لامتغير الحلقة الذي تصونه هذه الخوارزمية؟ لماذا يجب أن تنفذ فقط من أجل الـ $n-1$ عنصرًا الأولى من الصفيفة؟ أعط أزمنة تنفيذ الفرز الانتقائي في أفضل الحالات، وفي أسوأ الحالات باستخدام تدوين-Θ.

3-2.2

ليكن لدينا البحث الخطي ثانية (انظر التمرين 2.1-3). ما هو عدد العناصر التي يجب اختبارها وسطيًّا من متتالية الدخل، بافتراض أن العنصر الذي نبحث عنه قد يكون أيًّا من عناصر في الصفيفة باحتمالٍ متساوٍ؟ وما هو عدد العناصر التي يجب اختبارها في أسوأ الحالات؟ ما هو زمن تنفيذ البحث الخطي في الحالة الوسطى، وفي أسوأ الحالات باستخدام تدوين-Θ؟ علِّل أجوبتك.

4-2.2

كيف يمكننا تعديل – تقريبًا – أية خوارزمية للحصول على زمن تنفيذ جيد للحالة الفضلى؟

3.2 تصميم الخوارزميات

نستطيع الاختيار ضمن طيفٍ واسعٍ من تقنيات تصميم الخوارزميات. فقد استخدمنا في حالة خوارزمية الفرز بالإدراج طريقة ***تزايدية incremental***: فبعد فرز الصفيفة الجزئية $A[1..j-1]$، أدرجنا العنصر $A[j]$ في مكانه المناسب، لتنتج الصفيفة الجزئية المفروزة $A[1..j]$.

سنتطرق في هذا المقطع إلى طريقة تصميمٍ بديلة تدعى "فَرِّقْ-تَسُدْ divide-and-conquer"، سنعرضها بتفصيل أكبر في الفصل 4. سنستخدم مفهوم فرق-تسد لتصميم خوارزمية فرز زمن تنفيذها في أسوأ الحالات أقل كثيرًا من زمن تنفيذ الفرز بالإدراج في أسوأ الحالات. إحدى فوائد خوارزميات فرق-تسد أنه يمكن غالبًا تحديد أزمنة تنفيذها بسهولة باستخدام تقنيات سنراها في الفصل 4.

1.3.2 طريقة فَرِّق-تَسُدْ

كثير من الخوارزميات المفيدة ***عَوْدية recursive*** في بنيتها: فلحلّ مسألةٍ معطاة، تَستدعي هذه الخوارزميات نفسَها عَوْديًّا مرة أو أكثر لتعالج مسائلَ جزئية ذات علاقة وثيقة بالمسألة الأصلية. تَتَّبعُ هذه الخوارزميات في أغلب الأحيان طريقة ***فَرِّقْ-تَسُدْ divide-and-conquer***: حيث تُقَسِّمُ المسألة الأصلية إلى عدة مسائل جزئية تشابه المسألة الأصلية لكنها أصغر حجمًا، ثم تَحُلُّ هذه المسائلَ الجزئية عَوْديًّا، ثم تُجَمِّعُ هذه الحلول لتكوين حل المسألة الأصلية.

يحتوي نموذج فَرِّقْ-تَسُدْ على ثلاث خطوات في كل مستوى من العَوْدية:

فَرِّقْ: قَسِّمْ المسألة إلى عدد من المسائل الجزئية التي هي منتسخاتٌ instances أصغر من المسألة نفسها.

سُدْ: سَيْطِرْ على المسائل الجزئية بحلِّها عَوْديًّا. فإذا أصبحت حجومُ المسائل الجزئية صغيرةً كفايةً، فَحُلَّ المسائلَ الجزئيةَ مباشرة.

جَمِّعْ: جَمِّعْ حلولَ المسائل الجزئية لتكوين حلّ المسألة الأصلية.

تَتْبَعُ خوارزمية ***الفرز بالدمج Merge Sort*** مبدأ فَرِّقْ-تَسُدْ تمامًا؛ فهذه الخوارزمية تعمل ببساطة كما يلي:

فَرِّقْ: قَسِّمْ المتتاليةَ المكونةَ من n عنصرًا المطلوب فرزها إلى متتاليتين جزئيتين، يتكوَّن كل منهما من $n/2$ عنصرًا.

سُدْ: افْرِزْ المتتاليتين الجزئيتين عَوْديًّا باستخدام الفرز بالدمج.

جَمِّعْ: ادْمُجْ المتتاليتين الجزئيتين المفروزتين لإنتاج الجواب المفروز.

تنتهي العملية العَوْدية عندما يصبح طول المتتالية المطلوب فرزها يساوي 1، حيث لا يوجد ما يجب فعله في هذه الحالة، لأن كل متتالية طولها 1 تكون بترتيب مفروز حُكْمًا.

العملية الأساسية في خوارزمية الفرز بالدمج هي دمج متتاليتين مفروزتين في خطوة "التجميع". ندمج باستدعاء إجراء مساعد $\text{MERGE}(A,p,q,r)$ حيث A صفيفة، و p، و q، و r دلائل في الصفيفة بحيث يكون $p \le q < r$. يَفترض الإجراءُ أن الصفيفتين الجزئيتين $A[p\,..q]$ و $A[q+1\,..r]$ مفروزتان. ***يدمجهما merges*** هذا الإجراءُ ليكوِّن صفيفةً جزئية وحيدة مفروزة تحل محل الصفيفة الجزئية الحالية $A[p\,..r]$.

يستغرق الإجراء MERGE زمنًا $\Theta(n)$، حيث $n = r - p + 1$ هو عدد العناصر الكلّي التي يجري دمجها، ويعمل الإجراء كما يلي: بالعودة إلى تصورنا عن لعبة ورق الشدة، ليكن لدينا على الطاولة كومتين من أوراق الشدة وجوهها إلى أعلى. كل كومة مفروزة، بحيث تكون أصغر الأوراق في أعلى الكومة. نريد دمج هاتين الكومتين في كومة خرج واحدة مفروزة، بحيث تكون وجوه الأوراق فيها مقلوبة للأسفل على الطاولة. تتألف خطوتنا الأساسية من اختيار أصغر الورقتين الموضوعتين على قمتي كومتي الأوراق التي وجوهها إلى الأعلى، ثم إزالتها من كومتها (وهذا يؤدي إلى كشف ورقة قمة جديدة)، ووضع هذه الورقة على كومة الخرج ووجهها مقلوبٌ للأسفل. نكرِّر هذه الخطوة حتى تصبح إحدى كومتي الدخل فارغة، عندها فقط نأخذ كومة الدخل المتبقية ونضعها على كومة الخرج ووجهها إلى الأسفل. حسابيًّا، تستغرق كلُّ خطوة أساسية زمنًا ثابتًا، لأننا نقوم فقط بمقارنة ورقتَي قمة. ولما كنا ننفذ n خطوةً أساسية على الأكثر، فإن عملية الدمج تستغرق زمنًا $\Theta(n)$.

يُنَجِّزُ شبهُ الرماز التالي الفكرةَ المذكورة آنفًا، لكن مع تعديل إضافي يجنبنا الحاجة إلى التحقق من كون إحدى الكومتين فارغة في كل خطوة أساسية. نضع في أسفل كل كومةٍ ورقةً خاصة نسميها ***الورقة الكاشفة sentinel card***، تحتوي قيمة خاصة نستخدمها لتبسيط رمازنا. سنستخدم ∞ قيمةً كاشفةً، بحيث أنه حالما تنكشف الورقة ذات القيمة ∞، فلا يمكن أن تكون الورقة الصغرى ما لم تُظهر كلتا الكومتين ورقتيهما الكاشفتين. لكن ما إن يحدث ذلك، تكون جميع الأوراق الأخرى قد وُضعت على كومة الخرج. ولأننا نعرف سلفًا أنه ستوضع بالضبط $r - p + 1$ ورقة على كومة الخرج، فإننا نستطيع التوقف حالما نكون قد قمنا بهذا القدر من الخطوات الأساسية.

```
MERGE(A, p, q, r)
1  n1 = q - p + 1
2  n2 = r - q
3  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
4  for i = 1 to n1
5      L[i] = A[p + i - 1]
```

```
6   for j = 1 to n2
7       R[j] = A[q + j]
8   L[n1 + 1] = ∞
9   R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

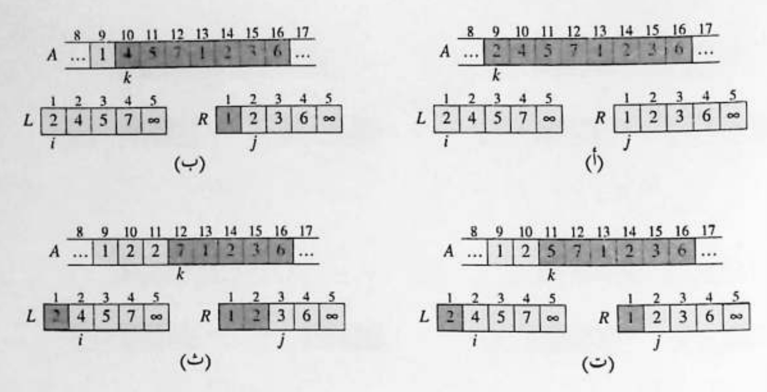
يعمل إجراء الدمج MERGE بالتفصيل كما يلي: يَحسب السطرُ الأول المقدارَ n_1 وهو طول الصفيفة الجزئية $A[p..q]$، ويحسب السطرُ الثاني المقدارَ n_2 وهو طول الصفيفة الجزئية $A[q+1..r]$. ننشئ في السطر الثالث الصفيفتين L و R ("الصفيفة اليسرى" و"الصفيفة اليمنى")، طول الأولى n_1+1، وطول الثانية n_2+1؛ وسيحتوي الموقع الإضافي في كل صفيفة الورقة الكاشفة. تَنسخ حلقةُ **for** المكونة من السطرين 4-5 الصفيفةَ الجزئية $A[p..q]$ في $L[1..n_1]$، وتنسخ حلقة **for** المكونة من السطرين 6-7 الصفيفةَ الجزئية $A[q+1..r]$ في $R[1..n_2]$. يَضَعُ السطران 8-9 الورقتين الكاشفتين في نهايتي الصفيفتين L و R. تُنجِزُ الأسطر 10-17، الموضحة في الشكل 3.2، الـ $r-p+1$ خطوة أساسية بالمحافظة على لامتغير الحلقة التالي:

عند بداية كل تكرار من حلقة **for**، التي تشمل الأسطر 12–17، تحتوي الصفيفة الجزئية $A[p..k-1]$ أصغر $k-p$ عنصرًا من $L[1..n_1+1]$ و $R[1..n_2+1]$، بترتيب مفروز. إضافةً إلى ذلك، يكون العنصران $L[i]$ و $R[j]$ أصغرَ عنصرين في صفيفتيهما لم ينسخا إلى A.

يجب أن نُثْبت أنَّ لامتغير الحلقة هذا محقَّق قبل التكرار الأول من حلقة **for** التي تشمل الأسطر 12-17، وأن كل تكرار من الحلقة يحافظ على اللامتغير، وأنَّ لامتغير الحلقة هذا يوفر خاصية مفيدة لإظهار صحة العمل عند توقف الحلقة.

الاستبداء: لدينا $k=p$ قبل التكرار الأول للحلقة، أي إن الصفيفة الجزئية $A[p..k-1]$ فارغة. تحتوي هذه الصفيفة الجزئية الفارغة أصغر $k-p=0$ عنصرًا من L و R، ولما كان $i=j=1$، فإن كلاً من العنصرين $L[i]$ و $R[j]$ هما أصغر عنصرين في صفيفتيهما لم يُعَدْ نسخهما في A.

المحافظة: حتى نرى أن كل تكرار يحافظ على لامتغير الحلقة، لنفترض أولاً أن $L[i] \le R[j]$. عندها يكون $L[i]$ هو أصغر عنصر لم يُعَدْ نَسْخه بَعْدُ إلى A. ولما كانت $A[p..k-1]$ تحتوي أصغر $k-p$ عنصرًا،



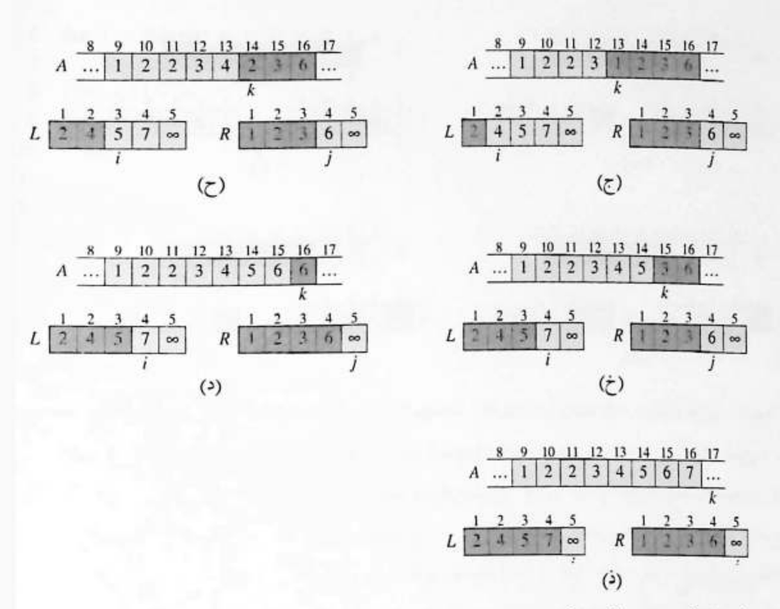
الشكل 3.2 عمل الأسطر 10-17 في الاستدعاء $\text{MERGE}(A,9,12,16)$، عندما تحتوي الصفيفة الجزئية $A[9..16]$ المتتالية $\langle 2,4,5,7,1,2,3,6\rangle$. بعد النسخ وإدخال القيمتين الكاشفتين، ستحتوي الصفيفة L القيم $\langle 2,4,5,7,\infty\rangle$، وستحتوي الصفيفة R القيم $\langle 1,2,3,6,\infty\rangle$. تحتوي المواضع المظللة باللون الفاتح في A قيمها النهائية، وتحتوي المواضع المظللة باللون الفاتح في L و R قيمًا يجب أن يعاد نسخها إلى A. بأخذ المواضع المظللة باللون الفاتح معًا، تحتوي هذه المواضع دائمًا القيم الموجودة أصلاً في $A[9..16]$، إضافة إلى القيمتين الكاشفتين. وتحتوي المواضع المظللة بكثافة في A قيمًا سوف يتم النسخ فوقها، وتحتوي المواضع المظللة بكثافة في L و R قيمًا تم إعادة نسخها إلى A. الأشكال من (أ) إلى (د) تمثل الصفيفات A و L و R التي دلائلها k و i و j على الترتيب، قبل كل تكرار من الحلقة في الأسطر 12-17.

فبعد أن يَنسخ السطر 14 العنصر $L[i]$ إلى $A[k]$، ستحتوي الصفيفة الجزئية $A[p..k]$ أصغر $k-p+1$ عنصرًا. تعيد زيادة كل من k (أثناء تحديث حلقة **for**) و i (في السطر 15) إعداد لامتغير الحلقة للتكرار التالي. أما في حال $L[i] > R[j]$، فإن السطرين 16-17 تقوم بما يجب للمحافظة على لامتغير الحلقة.

الإنهاء: عند الإنهاء، يكون $k = r+1$. واعتمادًا على لامتغير الحلقة، تحتوي الصفيفةُ الجزئية $A[p..k-1]$، التي هي $A[p..r]$، أصغرَ $k-p = r-p+1$ عنصرًا من $L[1..n_1+1]$ و $R[1..n_2+1]$، بترتيب مفروز. أي تحتوي الصفيفتان L و R معًا $n_1+n_2+2 = r-p+3$ عنصرًا. كلها قد أعيد نسخها في الصفيفة A، عدا أكبر عنصرين، وهما القيمتان الكاشفتان.

حتى ترى أن إجراء الدمج MERGE يُنفذ في زمن $\Theta(n)$، حيث $n = r-p+1$، لاحظ أن كلاً من الأسطر 1-3 و 8-11 يستغرق زمنًا ثابتًا، وتستغرق حلقات الـ **for** في الأسطر 4-7 زمنًا قدره $\Theta(n_1+n_2) = \Theta(n)$[7]، وأن هناك n تكرارًا لحلقة **for** في الأسطر 12-17، يستغرق كل تكرار منها زمنًا ثابتًا.

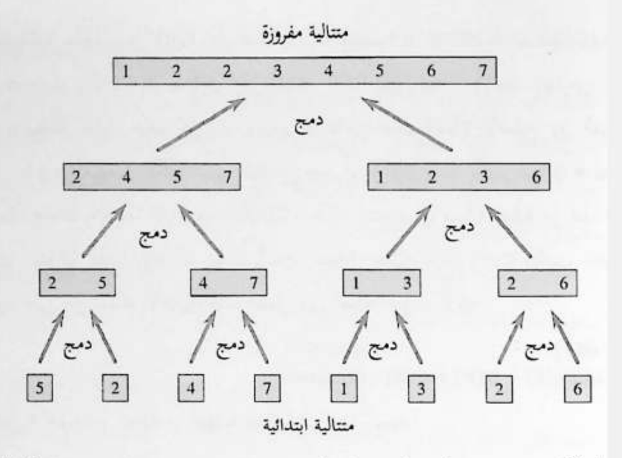
[7] سنرى في الفصل الثالث كيف تفسر المعادلات التي تحتوي على تدوين Θ صوريًا.



يَتْبَع الشكل 3.2 يمثّل الشكل (ذ) الصفيفات والمؤشرات عند الانتهاء. عند هذه المرحلة، تصبح الصفيفة الجزئية $A[9..16]$ مفروزة، والقيمتان الكاشفتان في L و R هما العنصران الوحيدان في هاتين الصفيفتين اللذان لم ينسخا بَعْدُ إلى A.

نستطيع الآن استخدام إجراء الدمج MERGE كمساق فرعي في خوارزمية الفرز بالدمج. يفرز الإجراء MERGE-SORT(A,p,r) عناصر الصفيفة الجزئية $A[p..r]$. إذا كان $p \geq r$، تحتوي الصفيفة الجزئية على الأكثر عنصرًا واحدًا وتكون مفروزة أصلاً. وفي الحالات الأخرى، تَحسب خطوةُ التقسيم ببساطة دليلاً q يقسم $A[p..r]$ إلى صفيفتين فرعيتين: الأولى: $A[p..q]$، وهي تحوي $\lceil n/2 \rceil$ عنصرًا، والثانية: $A[q+1..r]$، وهي تحوي $\lfloor n/2 \rfloor$ عنصرًا.[8]

[8] تعني العبارة $\lceil x \rceil$ أصغر عدد صحيح أكبر من x أو يساويه، وتعني العبارة $\lfloor x \rfloor$ أكبر عدد صحيح أقل من x أو يساويه. هذه التدوينات معرفة في الفصل 3. إن أسهل طريقة للتحقق من أن إعطاء q القيمة $\lfloor (p+r)/2 \rfloor$ ينتج الصفيفتين الفرعيتين $A[p..q]$ و $A[q+1..r]$ وطولهما $\lceil n/2 \rceil$ و $\lfloor n/2 \rfloor$ على الترتيب، هو اختبار الحالات الأربع التي تظهر تبعًا لكون كل من p و r فرديًا أو زوجيًا.



الشكل 4.2 عمل الفرز بالدمج على الصفيفة $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. تزداد أطوال المتتاليات المفروزة التي يتم دمجها كلما تقدمت الخوارزمية من الأسفل إلى الأعلى.

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

لفرز كامل المتتالية $A = \langle A[1], A[2], \dots, A[n] \rangle$ نقوم بالاستدعاء البدئي $\text{MERGE-SORT}(A, 1, A.length)$، حيث $A.length = n$. يوضح الشكل 4.2 كيفية عمل الإجراء من الأسفل إلى الأعلى عندما تكون n من قوى 2. تنطوي الخوارزمية على دمج أزواج من متتاليات ذات عنصر وحيد لتكوّن متتاليات مفروزة طول كل منها 2، ثم على دمج أزواج من المتتاليات ذات الطول 2 لتكوّن متتاليات مفروزة طول كل منها 4، و هكذا حتى تُدمَج متتاليتان طول كل منهما $n/2$ لتكوّنا المتتالية النهائية المفروزة وطولها n.

2.3.2 تحليل خوارزميات فَرِّقْ–تَسُدْ

عندما تحتوي خوارزمية ما استدعاءً عَوْديًّا، يكون بمقدورنا غالبًا وصف زمن تنفيذها ***بمعادلة عَوْدية recurrence equation*** أو ***بعلاقة عَوْدية recurrence***، تصف الزمن الكلي لمسألة من الحجم n بدلالة زمن التنفيذ على مدخلات أصغر. نستطيع عندها استخدام أدوات رياضية لحل العلاقة العَوْدية وإعطاء حدود لأداء الخوارزمية.

تنتج العلاقة العَوْدية لزمن تنفيذ خوارزمية فَرِّقْ–تَسُدْ من الخطوات الثلاث التي تكوّن إطار العمل

الأساسي. كما ذكرنا سابقًا، نجعل $T(n)$ زمن التنفيذ لمسألة حجمها n. إذا كان حجم المسألة صغيرًا كفايةً، وليكن $n \leq c$ حيث c ثابت ما، يستغرق الحل المباشر زمنًا ثابتًا، نكتبه كـ $\Theta(1)$. لنفترض أن تقسيمنا للمسألة يُنتِج a مسألةً جزئية، حجم كلٍّ منها يساوي $1/b$ من حجم المسألة الأصلية. (في الفرز بالدمج، يساوي كلٌّ من a و b القيمة 2، لكننا سنرى كثيرًا من خوارزميات فرق-تسد يكون فيها $a \neq b$.) يستغرق حل مسألة جزئية واحدة حجمها n/b زمنًا $T(n/b)$، وهكذا تستغرق a مسألة جزئية من هذا الحجم زمنًا $aT(n/b)$. إذا استغرقنا زمنًا $D(n)$ لتقسيم المسألة إلى مسائل جزئية، وزمنًا $C(n)$ لتركيب حلول المسائل الجزئية للحصول على حل المسألة الأصلية، فإننا نحصل على العلاقة العَوْدية الآتية:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c\,, \\ aT(n/b) + D(n) + C(n) & \text{otherwise}\,. \end{cases}$$

سنرى في الفصل 4 كيفية حل العلاقات العَوْدية الشائعة من هذه الصيغة.

تحليل الفرز بالدمج

على الرغم من أن شبه رماز خوارزمية MERGE-SORT يعمل على الوجه الصحيح عندما لا يكون عدد العناصر زوجيًّا، فإنه يمكن تبسيط تحليلنا المبني على العلاقة العَوْدية إذا افترضنا أن حجم المسألة الأصلية هو من قوى 2. عندها تُنتِج كلُّ خطوةِ تقسيمٍ متتاليتين جزئيتين حجم كل منهما $n/2$ تمامًا. سنرى في الفصل 4، أن هذه الفرضية لا تؤثر على مرتبة نمو حل العلاقة العَوْدية.

سنتبع المحاكمة المنطقية التالية لاستنتاج العلاقة العَوْدية التي يحققها $T(n)$، زمن تنفيذ خوارزمية الفرز بالدمج على n عددًا في أسوأ الحالات. يستغرق تنفيذ الفرز بالدمج على عنصرٍ واحدٍ فقط زمنًا ثابتًا. وعندما يكون لدينا عدد العناصر $n > 1$، نُجَزِّئُ زمن التنفيذ كما يلي:

فرِّقْ: تَحسب خطوةُ التفريقِ منتصفَ الصفيفة الجزئية فقط، وهذا يستغرق زمنًا ثابتًا. ومن ثم يكون: $D(n) = \Theta(1)$.

سُدْ: نحل عَوْديًّا مسألتين فرعيتين، حجمُ كلٍّ منهما $n/2$، فيسهم هذا العمل في زمن التنفيذ بمقدار $2T(n/2)$.

جَمِّعْ: لاحظنا توًّا أن إجراء الدمج MERGE المطبق على صفيفة جزئية حجمها n عنصرًا يستغرق زمنًا $\Theta(n)$، وبذلك يكون $C(n) = \Theta(n)$.

عندما نَجمع الدالتين $D(n)$ و $C(n)$ في تحليل الفرز بالدمج، فإننا نَجمع دالةً نموه $\Theta(n)$ مع دالة نموه $\Theta(1)$. إن هذا المجموع هو دالة خطية في n، أي $\Theta(n)$. بإضافة هذه الدالة إلى الحد $2T(n/2)$ الناتج من خطوة "السيادة conquer" تَنتج العلاقة العَوْدية لـ $T(n)$ زمن تنفيذ الفرز بالدمج في أسوأ الحالات:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \,. \end{cases} \tag{1.2}$$

سنرى، في الفصل 4، "المبرهنة الرئيسة master theorem"، التي يمكننا استخدامها لبيان أن $T(n)$ يساوي $\Theta(n \lg n)$، حيث $\lg n$ هو $\log_2 n$. ولما كان نمو الدالة اللغاريتمية أكثر بطئًا من أية دالة خطية، فإنه في حالة مدخلات كبيرة كفاية، يتفوق الفرز بالدمج ذي زمن التنفيذ $\Theta(n \lg n)$ على الفرز بالإدراج الذي زمن تنفيذه $\Theta(n^2)$، في أسوأ الحالات.

لا نحتاج إلى المبرهنة الرئيسة لنفهم بالحدس لماذا حل العلاقة العَوْدية (1.2) هو $T(n) = \Theta(n \lg n)$. سنعيد كتابة العلاقة العَوْدية (1.2) كما يلي:

$$T(n) = \begin{cases} c & \text{if } n = 1 \,, \\ 2T(n/2) + cn & \text{if } n > 1 \,. \end{cases} \tag{2.2}$$

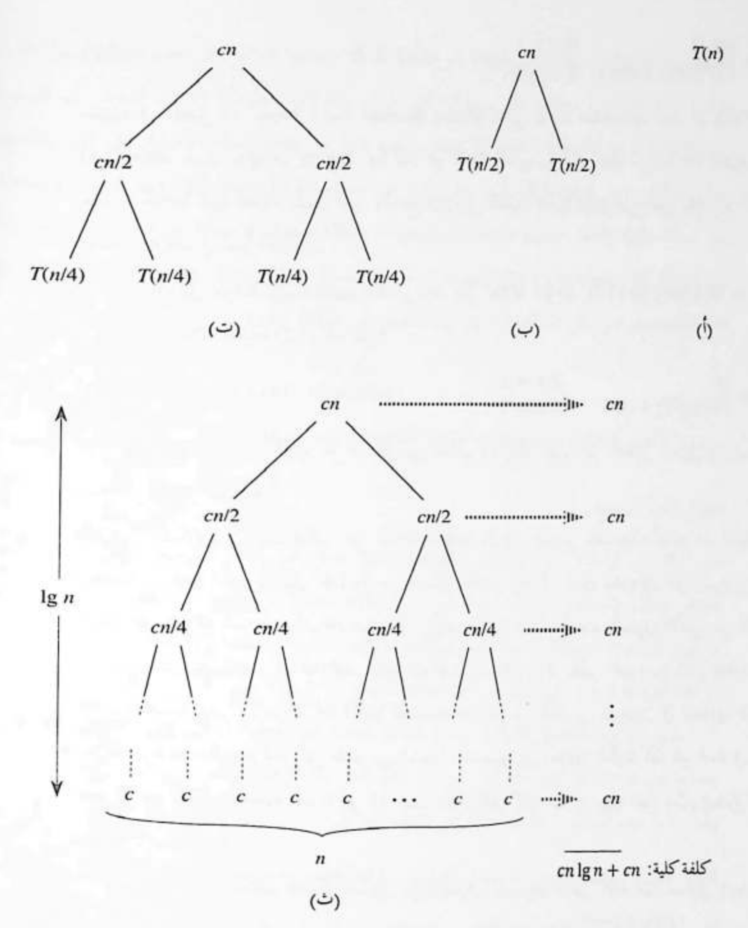
حيث يمثل الثابت c الزمن اللازم لحل مسائل من الحجم 1، ويمثل أيضًا زمن خطوتي التفريق والتجميع لكل عنصر من عناصر الصفيفة.[9]

يبين الشكل 5.2 كيف يمكننا حل العلاقة العَوْدية (2.2). نفترض للتبسيط، أن n هو بالضبط قوة صحيحة من قوى العدد 2. يبين الجزءُ (أ) من الشكل المقدارَ $T(n)$، الذي ننشره في الجزء (ب) إلى شجرة مكافئة تمثل العلاقة العَوْدية. الحد cn هو الجذر (وهو الكلفة المتضمنة عند المستوى الأعلى من العَوْدية)، والشجرتان الفرعيتان للجذر هما العلاقتان العَوْديتان الصغريان $T(n/2)$. يظهر الجزء (ت) إنجاز خطوة إضافية من هذه العملية بنشر $T(n/2)$. أما الكلفة المتضمنة عند كل من العقدتين الفرعيتين في المستوى الثاني من العَوْدية فهي $cn/2$. نتابع نشر كل عقدة من الشجرة بتقسيمها إلى أجزائها المكونة كما هو محدد في العلاقة العَوْدية، حتى تنخفض حجوم المسألة إلى 1، حيث تكون كلفة كل منها c. يبين الجزء (ت) ***شجرة العَوْدية recursion tree*** الناتجة.

بعد ذلك، نجمع الكلف عند كل مستوى من الشجرة. للمستوى الأعلى كلفة كلية تساوي cn، والكلفة الإجمالية للمستوى الأدنى التالي هي $c(n/2) + c(n/2) = cn$، والكلفة الكلية للمستوى الذي يليه هي $c(n/4) + c(n/4) + c(n/4) + c(n/4) = cn$، وهكذا دواليك. عمومًا، يمتلك المستوى i الأدنى بدءًا من قمة الشجرة 2^i عقدة، وتشارك كل عقدة منها بمقدار $c(n/2^i)$ من الكلفة، بحيث يكون للمستوى i الأدنى بدءًا من القمة كلفة كلية تساوي $2^i c(n/2^i) = cn$. يمتلك المستوى الأدنى الأخير n عقدة، تشارك

---

[9] من غير المحتمل أن يمثّل الثابت نفسه بدقة كلاً من زمن حل مسائل من الحجم 1 وزمن عنصر الصفيفة لخطوات التفريق والتجميع. يمكن أن نتجنب هذه المشكلة إما بأن نجعل c أكبر هذه الأزمنة، مع إدراكنا أن تكرار العَوْدية يعطي حدًّا أعلى لزمن التنفيذ، وإما بأن نجعل c أقل هذه الأزمنة، مع إدراكنا أن تكرار العَوْدية يعطي الحد الأدنى لزمن التنفيذ. على أن كلا الحدين سيكون $n \lg n$، وبأخذهما معًا يكون زمن التنفيذ $\Theta(n \lg n)$.



الشكل 5.2 كيفية بناء شجرة العَوْدية للعلاقة العَوْدية $T(n) = 2T(n/2) + cn$. يبين الجزء **(أ)** $T(n)$، الذي يمتد تدريجيًا في الأجزاء **(ب)**–**(ث)** لبناء شجرة العَوْدية. تمتلك شجرة العَوْدية المنشورة كاملةً في الجزء **(ث)** $\lg n + 1$ مستوًى (أي ارتفاعها $\lg n$، كما هو مشار إليه)، ويشارك كل مستوًى في الكلفة الكلية بالمقدار cn. وبناء على ذلك تساوي كلفة الزمن الكلية المقدار $cn \lg n + cn$، التي هي $\Theta(n \lg n)$.

كل منها في الكلفة بمقدار c، وتكون الكلفة الكلية لهذا المستوى هي cn.

إن العدد الكلي لمستويات الشجرة العَوْدية recursion tree في الشكل 5.2 هو $\lg n + 1$، حيث n عدد الأوراق، وهو يقابل حجم الدخل. ويمكن برهان هذا الادعاء بمحاكمة استقرائية مبسطة. تحدث الحالة

الأساسية عندما تكون $n = 1$، في هذه الحالة تمتلك الشجرة مستوًى واحدًا فقط. ولما كان $\lg 1 = 0$، فإن $\lg n + 1$ يعطي عدد المستويات الصحيح. لنفترض الآن فرضية استقرائية هي أن عدد مستويات الشجرة العَوْدية لـ 2^i ورقة هو $\lg 2^i + 1 = i + 1$ (لأنه في حالة أية قيمة i نحصل على $\lg 2^i = i$). وحيث إننا افترضنا أن حجم الدخل هو أحد قوى العدد 2، فإن حجم الدخل التالي هو 2^{i+1}. تمتلك الشجرة التي تحتوي $n = 2^{i+1}$ ورقة مستوى إضافيًا واحدًا مقارنة بعدد المستويات التي تمتلكها الشجرة التي عدد أوراقها 2^i، وهكذا يكون العدد الكلي للمستويات هو $(i + 1) + 1 = \lg 2^{i+1} + 1$.

لحساب الكلفة الكلية الممثَّلة بالعلاقة العَوْدية (2.2)، نجمع ببساطة كلف جميع المستويات من الأسفل إلى الأعلى. ولما كانت شجرة العَوْدية تمتلك $\lg n + 1$ مستوًى، كلفةُ كلٍّ منها cn، فالكلفة الكلية تساوي $cn(\lg n + 1) = cn \lg n + cn$. وبتجاهل الحد ذي الدرجة الأدنى والثابت c نحصل على النتيجة المطلوبة وهي: $\Theta(n \lg n)$.

تمارين

1-3.2

باستخدام الشكل 4.2 نموذجًا، وضِّح عمل الفرز بالدمج على الصفيفة التالية: $A = \langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$.

2-3.2

أعد كتابة إجراء الـ MERGE بحيث لا يستخدم الأوراق الكاشفة، واجعله بدلاً عن ذلك يتوقف بمجرد أن تكون أيٌّ من الصفيفتين L أو R قد أعادت نسخ جميع عناصرها إلى الصفيفة A، ثم يقوم بإعادة نسخ العناصر المتبقية من الصفيفة الأخرى إلى A.

3-3.2

استخدم الاستقراء الرياضي لإظهار أنه عندما تكون n قوة صحيحة للعدد 2، فإن حل العلاقة العَوْدية التالية:

$$T(n) = \begin{cases} 2 & \text{if } n = 1\,, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

هو $T(n) = n \lg n$.

4-3.2

يمكننا التعبير عن الفرز بالإدراج كإجراء عَوْدي كما يلي: لفرز $A[1..n]$، نفرز عَوْديًّا $A[1..n-1]$، ثم نُدرج $A[n]$ في الصفيفة المرتبة $A[1..n-1]$. اكتب العلاقة العَوْدية لزمن التنفيذ في أسوأ الحالات لهذه النسخة العَوْدية من الفرز بالإدراج.

5-3.2

بالرجوع إلى مسألة البحث (انظر التمرين 2.1-3)، لاحظ أنه إذا كانت المتتالية A مفروزة، نستطيع اختبار نقطة وسط هذه المتتالية مقارنة بالقيمة v وإخراج نصف المتتالية من دائرة بحثنا التالي. تكرّر خوارزميةُ ***البحث الثنائي binary search*** هذا الإجراء، وبذلك تنصّف حجم الجزء الباقي من المتتالية في كل مرة. اكتب شبه رماز للبحث الثنائي، إما تكراريًا وإما عَوْديًا. بيِّن أن زمن تنفيذ البحث الثنائي في أسوأ الحالات هو $\Theta(\lg n)$.

6-3.2

لاحظ أن حلقة **while** المكونة من الأسطر 5-7 من إجراء INSERTION-SORT في المقطع 1.2 تستخدم البحث الخطي لمسح الصفيفة الجزئية المفروزة $A[1..j-1]$ عكسيًا. هل نستطيع استخدام البحث الثنائي (انظر التمرين 2.3-5) عوضًا عن تحسين زمن تنفيذ الفرز بالإدراج الكلي في أسوأ الحالات ليصبح $\Theta(n \lg n)$؟

★ *7-3.2*

صِفْ خوارزميةً ذات زمن تنفيذ قدره $\Theta(n \lg n)$ تقوم – عند إعطائها مجموعةً S مؤلَّفةً من n عددًا صحيحًا وعددٍ صحيحٍ آخر x – بتحديد وجودِ (أو عدم وجود) عنصرين في S مجموعهما هو x تمامًا.

مسائل

2-1 الفرز بالإدراج على صفيفات صغيرة داخل الفرز بالدمج

على الرغم من أن زمن تنفيذ الفرز بالدمج في أسوأ الحالات هو $\Theta(n \lg n)$، وزمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو $\Theta(n^2)$، فإن العوامل الثابتة في الفرز بالإدراج يمكن أن تجعله أسرع عمليًا في حالة حجوم المسائل الصغيرة على العديد من الحواسيب. وهذا ما قد يجعل مبررًا ***تسميك coarsen*** الأوراق في العَوْدية باستخدام الفرز بالإدراج ضمن الفرز بالدمج عندما تصبح المسائل الجزئية صغيرة كفاية. لندرس تعديلاً للفرز بالدمج تُفرَز فيه لوائح عددها n/k، طول كل منها k باستخدام الفرز بالإدراج، ومن ثم تدمج باستخدام آلية الدمج المعيارية، حيث k قيمة يجب تحديدها.

أ. بيِّن أنه يمكن للفرز بالإدراج أن يفرز n/k لائحة جزئية، طول كل منها k، في زمن $\Theta(nk)$ في أسوأ الحالات.

ب. بيِّن كيف يمكن دمج اللوائح الجزئية في زمن $\Theta(n \lg(n/k))$ في أسوأ الحالات.

ت. إذا علمنا أن زمن تنفيذ الخوارزمية المعدَّلة في أسوأ الحالات هو $\Theta(nk + n \lg(n/k))$، ما هي أكبر

قيمة لـ k بصفتها دالة لـ n يكون زمن تنفيذ الخوارزمية المعدلة عندها مماثلاً لزمن تنفيذ خوارزمية الفرز بالدمج المعيارية، باستخدام تدوين-Θ؟

ث. كيف يجب أن نختار k عمليًا؟

2-2 صحة الفرز الفقاعي

الفرز الفقاعي bubblesort، خوارزمية فرز شائعة، لكنها غير فعالة، تعمل على التبديل المتكرر للعناصر المتجاورة غير المرتبة.

```
BUBBLESORT(A)
1  for i = 1 to A.length - 1
2      for j = A.length downto i + 1
3          if A[j] < A[j - 1]
4              exchange A[j] with A[j - 1]
```

أ. لتكن A' الصفيفة التي تشير إلى خرج BUBBLESORT(A). حتى نثبت أن إجراء BUBBLESORT صحيح، نحتاج إلى إثبات أنه يتوقف، وأن:

$$A'[1] \leq A'[2] \leq \cdots \leq A'[n] \, , \qquad (3.2)$$

حيث $n = A.length$. ما الذي يجب أن نُثْبِتَه أيضًا لنبيّن أن BUBBLESORT يَفْرُز فعلاً؟

سيثبت الجزءان التاليان صحة المتراجحة (3.2).

ب. أعطِ بدقة لامتغير حلقة **for** في الأسطر 2-4، وأثبت صحته. يجب أن يستخدم برهانك بنية برهان لامتغير الحلقة الذي عُرِضَ في هذا الفصل.

ت. باستخدام شرط توقف لامتغير الحلقة المبَرْهَن في الجزء (ب)، ضع لامتغير حلقة لحلقة **for** في الأسطر 1-4 يسمح لك ببرهان المتراجحة (3.2). يجب أن يستخدم برهانك بنية برهان لامتغير الحلقة الذي عُرِضَ في هذا الفصل.

ث. ما هو زمن تنفيذ الفرز الفقاعي في أسوأ الحالات؟ قارنه بزمن تنفيذ الفرز بالإدراج؟

3-2 صحة قاعدة هورنر

يُنَجِّزُ مقطع الرماز التالي قاعدة هورنر Horner's rule المستخدمة لتقييم كثير حدود

$$\begin{aligned} P(x) &= \sum_{k=0}^{n} a_k x^k \\ &= a_0 + x(a_1 + x(a_2 \ldots + x(a_{n-1} + x a_n) \ldots)) \, , \end{aligned}$$

ولنكن لدينا قيم المعاملات $a_0, a_1, \dots, a_n$ وقيمة x معطاة:

```
1  y = 0
2  for i = n downto 0
3       y = a_i + x . y
```

أ. ما هو زمن تنفيذ قطعة الرماز السابقة المقابلة لقاعدة هورنر، باستخدام تدوين-Θ ؟

ب. اكتب شبه رماز لتنجيز خوارزمية بسيطة لتقييم كثير حدود تَحْسِبُ كلَّ حدٍّ من كثير الحدود بدءًا من البداية. ما هو زمن تنفيذ هذه الخوارزمية؟ قارِنْه بزمن تنفيذ قاعدة هورنر؟

ت. ليكن لدينا لامتغير الحلقة التالي:

في بداية كل تكرار لحلقة **for** التي تشمل السطرين 2-3،

$$y = \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^k \ .$$

فسّر مجموعًا خاليًا من الحدود على أنه يساوي 0. باتباع بنية برهان لامتغير الحلقة الذي عُرِضَ في هذا الفصل، استخدم لامتغير الحلقة هذا لإظهار أن $y = \sum_{k=0}^{n} a_k x^k$ عند التوقف.

ث. اختتم عملك بمناقشة أن قطعة الرماز المعطى تحسب -على نحو صحيح- كثيرَ حدودٍ موصوفًا بالمعاملات $a_0, a_1, \dots, a_n$.

4-2 *العكوس*

لتكن $A[1..n]$ صفيفةً مؤلَّفةً من n عددًا متمايزاً. إذا كان $i < j$ و $A[i] > A[j]$، عندها يسمَّى الزوج (i, j) ***عكسًا inversion*** في A.

أ. اذكرْ العكوس الخمسة الموجودة في الصفيفة $A = \langle 2, 3, 8, 6, 1 \rangle$.

ب. ما الصفيفة التي عناصرها من المجموعة $\{1, 2, \dots, n\}$ والتي تحتوي أكبر عدد من العكوس؟ وكم عكسًا تحتوي؟

ت. ما هي العلاقة بين زمن تنفيذ الفرز بالإدراج وعدد العكوس في صفيفة الدخل؟ علل جوابك.

ث. أعط خوارزمية تحدد عدد العكوس في أي تبديل على n عنصرًا، زمن تنفيذها في أسوأ الحالات هو $\Theta(n \lg n)$. (*تلميح*: عَدِّلْ الفرز بالدمج.)

ملاحظات الفصل

في عام 1968 نشر Knuth الجزء الأول من ثلاثة أجزاء من كتاب بعنوان عام هو فن برمجة الحاسوب *The Art of Computer Programming* [209, 210, 211]. قدّم الجزء الأول للطريقة الحديثة في دراسة خوارزميات الحاسوب التي تركّز على تحليل زمن التنفيذ. والسلسلة بكاملها مرجعٌ جذاب وقيم لكثير من المواضيع المعروضة هنا. تُشْتَقُّ كلمة "خوارزمية algorithm"، حسب Knuth، من اسم الخوارزمي، وهو عالم رياضيات فارسي عاش في القرن التاسع.

دافع Aho و Hopcroft و Ullman [5] عن التحليل المقارب للخوارزميات – باستخدام التدوينات التي يعرضها الفصل 3، ومنها تدوين-Θ – بصفتها طريقة لمقارنة الأداء النسبي. وقد أشاع هؤلاء المؤلفون استخدام العلاقات العَوْدية لوصف أزمنة تنفيذ الخوارزميات العَوْدية.

قدّم Knuth [211] معالجة موسوعية لكثير من خوارزميات الفرز. تحتوي مقارنته لخوارزميات الفرز (في الصفحة 381) تحليلات دقيقة تعد الخطوات مشابهة للتحليل الذي قمنا به هنا في حالة الفرز بالإدراج. تتضمن مناقشة Knuth للفرز بالإدراج عدة تعديلات على هذه الخوارزمية. أهمها خوارزمية فرز شيل Shell's sort، التي أدخلها D. L. Shell، والتي تستخدم الفرز بالإدراج على متتاليات جزئية دورية من الدخل لتعطي خوارزمية فرز أسرع.

وَصَفَ Knuth أيضًا خوارزمية الفرز بالدمج. وتحدَّث في كتابه عن آلة دمج ميكانيكية mechanical collator، يعود تاريخ اختراعها إلى 1938، كانت قادرة على دمج مجموعتين من البطاقات المثقبة بمرور واحد. ويبدو أن J. Von Neumann أحد الرواد في علوم الحاسوب، كان قد كتب برنامجًا للفرز بالدمج على حاسوب EDVAC عام 1945.

وَصَفَ Gries [153] البدايات المبكرة لبرهان صحة البرامج، ونَسَبَها إلى P. Naur صاحب أول مقال في هذا الحقل. ونَسَبَ Gries لامتغيرات الحلقة إلى R. W. Floyd. يشرح الكتاب الجامعي لـ Mitchell [256] أحدث التطورات التي طرأت على برهان صحة البرامج.

3 نُموُّ الدوالّ

تعطي مرتبة نمو زمن تنفيذ خوارزمية ما، كما عرّفناها في الفصل الثاني، توصيفًا بسيطًا لفعالية الخوارزمية، وتسمح لنا أيضًا بمقارنة أداء خوارزميات بديلة فيما بينها. فمثلاً، ما إن يصبح حجم المُدخلات n كبيرًا كفاية، حتى يتغلب الفرز بالدمج بزمن تنفيذه في أسوأ الحالات $\Theta(n \lg n)$ على الفرز بالإدراج ذي زمن التنفيذ في أسوأ الحالات $\Theta(n^2)$. ومع أننا نستطيع في بعض الأحيان تحديد زمن التنفيذ الدقيق لخوارزمية ما، كما فعلنا في الفرز بالإدراج في الفصل الثاني، إلا أن الدقة الزائدة لا تستحق عادة العناء المبذول للحصول عليها. ففي حالة مُدخلات كبيرة كفاية، يغطي حجم المُدخلات نفسه على ثوابت الجداء وعلى الحدود من المراتب الصغرى.

عندما ننظر إلى حجوم مُدخلات كبيرة كفاية تكون مَرْتبَةُ نمو زمن التنفيذ فقط ذات معنى، ونكون بصدد دراسة الفعالية ***المقاربة asymptotic*** للخوارزميات. أي إننا نهتم بكيفية تزايد زمن تنفيذ خوارزمية ما تبعًا لتزايد حجم المُدخلات *عند النهاية*، أي عندما يصبح حجم المُدخَلات غير محدود. وعادةً، تكون أكثر الخوارزميات فعاليّة بالمقاربة هي الخيار الأفضل لكل الحجوم ما عدا الحجوم الصغيرة جدًّا.

يقدّم هذا الفصل عدّة طرق قياسيّة لتبسيط التحليل المقارب للخوارزميات. ويبدأ المقطع التالي بتعريف عدة أنواع من "التدوين المقارب asymptotic notation" التي سبق أن رأينا مثالاً عنها، وهو تدوين–Θ. ثمّ نقدّم مجموعة من الاصطلاحات التدوينيّة المستخدمة في هذا الكتاب، وأخيرًا نراجع سلوك الدوال التي تَظهر غالبًا عند تحليل الخوارزميات.

1.3 التدوين المقارب

تعتمد التدوينات التي نستخدمها لوصف زمن التنفيذ المقارب لخوارزمية ما على دوال معرّفة على مجموعة الأعداد الطبيعيّة $\mathbb{N} = \{0, 1, 2, \ldots\}$. وهذه التدوينات مناسبة لوصف دالة زمن التنفيذ في أسوأ الحالات $T(n)$، التي تكون معرّفة عادة على حجوم المُدخلات الصحيحة فقط. ومع ذلك، فمن المناسب في بعض الأحيان، أن نقبل ببعض التجاوزات في التدوين المقارب؛ فمثلاً، يمكن بسهولة توسيع التدوين ليشمل حقل

الأعداد الحقيقية، أو على العكس حصره في مجموعة جزئية من الأعداد الطبيعيّة. إلا أنه من المهم أن نَعِيَ تمامًا معنى التدوين بحيث لا نسيء استخدامه بوجود هذه التجاوزات. يعرِّف هذا المقطع التدوينات المقاربة الأساسيّة، ويقدّم أيضًا بعض التجاوزات الشائعة.

التدوين المقارب، والدوال، وأزمنة التنفيذ

سنستخدم التدوين المقارب أساسًا لوصف أزمنة تنفيذ الخوارزميات، مثلما فعلنا عندما ذكرنا أن زمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو $\Theta(n^2)$، وذلك على الرغم من أن التدوين المقارب يُطبّق في الواقع على الدوال. تذكّر أننا وصّفنا زمن تنفيذ الفرز بالإدراج في أسوأ الحالات بالصيغة $an^2 + bn + c$، حيث a و b و c ثوابت. فإذا كتبنا زمن تنفيذ الفرز بالإدراج بالصيغة $\Theta(n^2)$، نكون قد أغفلنا بعض تفاصيل هذه الدالة. ولما كان التدوين المقارب يُطبَّق على الدوال، فإن ما كتبناه على أنه $\Theta(n^2)$ هو الدالة $an^2 + bn + c$، التي تصادف أنها توصّف زمن تنفيذ الفرز بالإدراج في أسوأ الحالات.

في هذا الكتاب، ستكون الدوال التي سنطبّق عليها التدوين المقارب في غالب الأحيان توصيفات لأزمنة تنفيذ خوارزميات. إلا أن التدوين المقارب يمكن أن ينطبق على دوال توصِّف جوانب أخرى من الخوارزميات (مثلاً، حجم الذاكرة الذي تشغله)، أو حتى على دوال لا علاقة لها البتة بالخوارزميات.

وحتى عندما نستخدم التدوين المقارب لتطبيقه على زمن تنفيذ خوارزمية ما، نحتاج إلى فهم *أيّ* زمنِ تنفيذٍ نَعْنِي. ففي بعض الأحيان، نهتم بزمن التنفيذ في أسوأ الحالات، إلا أننا كثيرًا ما نرغب في توصيف زمن التنفيذ مهما كان الدخل. وبعبارة أخرى، كثيرًا ما نرغب في إعطاء تصريح يشمل المدخلات كلها، وليس مدخلات أسوأ الحالات فحسب. وسنطَّلع على تدوينات مقاربة مناسبة لتوصيف أزمنة التنفيذ مهما كان الدخل.

تدوين-Θ

وجدنا في الفصل الثاني أن زمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو $T(n) = \Theta(n^2)$. سنعرّف مفهوم هذا التدوين لدالة معطاة $g(n)$، ونرمز له بـ $\Theta(g(n))$، بأنه مجموعة الدوال:

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that}$$
$$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}\,.$$[1]

تنتمي دالة $f(n)$ إلى المجموعة $\Theta(g(n))$ إذا كان هناك ثابتان c_1 و c_2 بحيث يمكن "حشر sandwiched" هذه الدالة بين $c_1 g(n)$ و $c_2 g(n)$، عندما تكون n كبيرة كفاية. ولما كانت $\Theta(g(n))$ مجموعة، كان

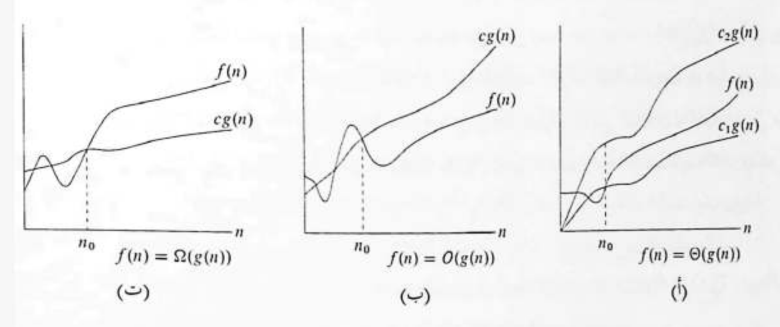
---

[1] النقطتان ":" تُقرأان في تدوين المجموعات "حيث".

بإمكاننا أن نكتب "$f(n) \in \Theta(g(n))$" لنبيّن أن $f(n)$ هي عنصر من $\Theta(g(n))$. ولكننا بدلاً من ذلك نكتب عادة "$f(n) = \Theta(g(n))$" للتعبير عن المفهوم نفسه. قد يبدو هذا التجاوز باستخدام المساواة عوضًا عن إشارة الانتماء في البداية مشوشًا بعض الشيء، ولكننا سنرى بعد قليل في هذا المقطع أن له فوائده.

يعطي الشكل 1.3 (أ) تمثيلاً مبسطًا للدوال $f(n)$ و $g(n)$، حيث $f(n) = \Theta(g(n))$. تقع قيمة $f(n)$ فوق $c_1g(n)$ وأسفل $c_2g(n)$، أيًّا كانت قيمة n الواقعة إلى يمين n_0. وبعبارة أخرى، أيًّا كانت $n \geq n_0$، فإن الدالة $f(n)$ تساوي $g(n)$ ضمن مُعامل ثابت. ونقول إن الدالة $g(n)$ هي ***حدّ مُحكم بالمقارنة asymptotically tight bound*** للدالة $f(n)$.

يتطلّب تعريف $\Theta(g(n))$ أن يكون كل عنصر $f(n) \in \Theta(g(n))$ ***موجبًا بالمقارنة asymptotically nonnegative***، أي أن تكون الدالة $f(n)$ موجبة عندما تصبح n كبيرة كفاية. (***الدالة الموجبة تمامًا بالمقارنة asymptotically positive*** هي دالة موجبة تمامًا لكل قيم n الكبيرة كفاية.) بالنتيجة، يجب أن تكون الدالة $g(n)$ نفسها موجبة بالمقارنة، وإلاّ فإن المجموعة $\Theta(g(n))$ تكون فارغة. ولذلك نفترض أن كل الدوال المستخدمة في تدوين-Θ هي موجبة بالمقارنة. وتبقى هذه الفرضيّة محققة أيضًا في بقية التدوينات المقاربة التي سنتعرّفها في هذا الفصل.



الشكل 1.3 أمثلة بيانية للتدوينات Θ، و O، و Ω. إن قيمة n_0 المبينة في كل جزء من الشكل هي أصغر قيمة ممكنة؛ ويمكن أن تحل محلها أية قيمة أكبر منها. (أ) يَحدّ تدوين-Θ دالة ما ضمن معاملين ثابتين. ونكتب $f(n) = \Theta(g(n))$ إذا وُجدت ثوابت موجبة n_0، و c_1، و c_2 بحيث، عندما تقع n عند n_0 وإلى يمينها، تقع قيمة $f(n)$ دائمًا بين $c_1g(n)$ و $c_2g(n)$ أو تساويهما. **(ب)** يعطي تدوين-O حدًّا أعلى لدالة ما ضمن معامل ثابت. ونكتب $f(n) = O(g(n))$ إذا وُجد ثابتان موجبان n_0، و c بحيث، عندما تقع n عند n_0 وإلى يمينها، تقع قيمة $f(n)$ دائمًا على $cg(n)$ أو تحته. **(ت)** يعطي تدوين-Ω حدًّا أدنى لدالة ما ضمن معامل ثابت. ونكتب $f(n) = \Omega(g(n))$ إذا وُجد ثابتان موجبان n_0، و c بحيث، عندما تقع n عند n_0 وإلى يمينها، تقع قيمة $f(n)$ دائمًا على $cg(n)$ أو فوقه.

قدّمنا في الفصل الثاني تفسيرًا تقريبيًّا لمفهوم تدوين-Θ يتمثّل في التخلّص من الحدود الأدنى مرتبة وفي تجاهل المعامل المرافق للحد الأعلى مرتبة. دعنا نبرّر بإيجاز هذا الحدس باستخدام التعريف الصوري لنبيّن أن $\frac{1}{2}n^2 - 3n = \Theta(n^2)$. وللقيام بذلك، يجب أن نحدّد الثوابت الموجبة c_1 و c_2 و n_0 بحيث يكون

$$c_1n^2 \le \frac{1}{2}n^2 - 3n \le c_2n^2$$

أيًّا كانت $n \ge n_0$. وبالتقسيم على n^2 نحصل على

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2 \, .$$

يمكن تحقيق المتراجحة اليمنى أيًّا كانت $n \ge 1$ باختيار $c_2 \ge 1/2$. وبالمثل يمكن تحقيق المتراجحة اليسرى أيًّا كانت $n \ge 7$ باختيار $c_1 \le 1/14$. وهكذا، وباختيار $c_1 = 1/14$ و $c_2 = 1/2$ و $n_0 = 7$، يمكن أن نتحقق من أنّ $\frac{1}{2}n^2 - 3n = \Theta(n^2)$. طبعًا يوجد خيارات أخرى لقيم الثوابت، ولكن ما يهمّ هنا هو وجود قيم ممكنة. لاحظ أن هذه الثوابت تتعلق بالدالة $\frac{1}{2}n^2 - 3n$؛ وأية دالة أخرى من $\Theta(n^2)$ تتطلب عادة ثوابت مختلفة.

بوسعنا أيضًا استخدام التعريف الصوري للتحقق من أن $6n^3 \ne \Theta(n^2)$. افترض، بهدف الوصول إلى تناقض، أنه يوجد c_2 و n_0 بحيث يكون $6n^3 \le c_2n^2$ أيًّا كانت $n \ge n_0$. ولكن بالتقسيم على n^2 نحصل على $n \le c_2/6$ والذي لا يمكن أن يتحقق عندما تكون n كبيرة كفاية، وذلك لأن c_2 ثابت.

إذن، يمكن بالحدس تجاهل الحدود الأدنى مرتبةً في دالة موجبة بالمقاربة عند تحديد حدود محكمة بالمقاربة، لأنها تكون غير ذات معنى في حالة قيم كبيرة لـ n. إن جزءًا صغيرًا من الحد الأعلى مرتبةً كافٍ ليفوق هذه الحدود الأدنى مرتبة. وهكذا، يسمح إعطاء قيمة لـ c_1 أصغر قليلاً من معامل الحد الأعلى مرتبة، وإعطاء قيمة لـ c_2 أكبر قليلاً من هذا المعامل، بتحقيق المتراجحات في تعريف التدوين-Θ. ويمكن تجاهل معامل الحد الأعلى مرتبة أيضًا، لأنه يغيّر فقط قيم c_1 و c_2 بعامل ثابت مساوٍ لهذا المعامل.

كمثال على ذلك، لنأخذ أية دالة تربيعيّة $f(n) = an^2 + bn + c$، حيث a و b و c ثوابت و $a > 0$. إن إهمال الحدود الأدنى مرتبة وتجاهل الثابت يعطينا $f(n) = \Theta(n^2)$. حتى نتوصل للنتيجة نفسها بطريقة صوريّة، نأخذ $c_1 = a/4$، و $c_2 = 7a/4$، و $n_0 = 2 \cdot \max\left(|b|/a, \sqrt{|c|/a}\right)$. يمكن للقارئ أن يتحقق من أنّ $0 \le c_1n^2 \le an^2 + bn + c \le c_2n^2$ أيًّا كانت $n \ge n_0$. وعمومًا في حالة أي كثير حدود $p(n) = \sum_{i=0}^{d} a_in^i$، حيث a_i ثوابت و $a_d > 0$، يكون لدينا $p(n) = \Theta(n^d)$ (انظر المسألة (3-1)).

ولما كان أي ثابت هو كثير حدود من الدرجة 0، فيمكن أن نعبّر عن أيّة دالة ثابتة بأنها $\Theta(n^0)$،

أو $\Theta(1)$. إن التدوين الثاني هو تجاوز بسيط، إذ لا يظهر فيه بوضوح المتحول الذي يسعى إلى اللانهاية.[2] سنستخدم التدوين $\Theta(1)$ مرارًا لنعني به إما ثابتًا أو دالة ثابتة بالنسبة لمتحول ما.

تدوين$-O$

إن التدوين$-\Theta$ يَحدّ دالةً ما بالمقاربة من الأعلى ومن الأسفل. عندما يكون لدينا ***حد أعلى بالمقاربة*** ***asymptotic upper bound*** فقط، فإننا نستخدم التدوين$-O$. فإذا كانت $g(n)$ دالةً معطاة، فإننا نعني بالعبارة $O(g(n))$ [والتي تلفظ "big-oh of g of n"، أو أحيانًا "oh of g of n" فقط] مجموعةَ الدوال

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that}$$
$$0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$$

نستخدم تدوين$-O$ لنعطي حدًّا أعلى لدالة ما مضروبًا بثابت، يبيّن الشكل 1.3(ب) الحدس الذي يرتكز عليه تدوين$-O$: لكل قيم n عند n_0 وإلى يمينها، تكون قيمة الدالة $f(n)$ مساوية لـ $cg(n)$ أو أقل منها.

نكتب $f(n) = O(g(n))$ لنشير إلى أن الدالة $f(n)$ هي عنصر من المجموعة $O(g(n))$. لاحظ أنّ $f(n) = \Theta(g(n))$ يقتضي أن $f(n) = O(g(n))$، لأن تدوين$-\Theta$ أقوى من تدوين$-O$. وباستخدام لغة نظرية المجموعات، يمكننا كتابة $\Theta(g(n)) \subseteq O(g(n))$. إذن، إن برهاننا بأن أية دالة تربيعية $an^2 + bn + c$ حيث $a > 0$، هي $\Theta(n^2)$، يبيّن أيضًا أن أية دالة تربيعيّة تنتمي إلى $O(n^2)$. وقد يكون مفاجئًا أكثر أنّ أية دالة خطّية $an + b$ عندما $a > 0$، تنتمي إلى $O(n^2)$، وهذا ما يمكن التحقق منه بسهولة بأخذ $c = a + |b|$ و $n_0 = \max(1, -b/a)$.

قد يجد بعض القراء الذين تعرّضوا لتدوين$-O$ من قبل غرابةً في أن نكتب مثلاً $n = O(n^2)$. يُستخدَم تدوين$-O$ في الأدبيات أحيانًا لوصف حدود محكمة بالمقاربة، وهذا ما عرّفناه باستخدام تدوين$-\Theta$. إلا أنه، عندما نكتب في هذا الكتاب $f(n) = O(g(n))$ فإننا ندّعي فقط أن هناك مضاعفًا ثابتًا لـ $g(n)$ هو حد أعلى بالمقاربة لـ $f(n)$، دون أيّة ادعاءات عن مدى إحكام هذا الحدّ الأعلى. إن التمييز بين الحدود العليا بالمقاربة والحدود المحكمة بالمقاربة أصبح الآن متعارفًا في أدبيات الخوارزميات.

يمكننا في كثير من الأحيان، باستخدام تدوين$-O$، وصف زمن تنفيذ إجرائية ما بتفحص البنية العامة للخوارزمية فقط. فعلى سبيل المثال، تعطي بنية الحلقتين المتداخلتين في خوارزمية الفرز بالإدراج – التي تعرّضنا لها في الفصل الثاني – مباشرة حدًّا أعلى لزمن التنفيذ في أسوأ الحالات هو $O(n^2)$، فكلفة كل تكرار للحلقة

---

[2] المشكلة الحقيقية هي أن التدوين المعتاد للدوال لا يميّز بين الدوال والقيم؛ ففي حساب$-\lambda$ λ-calculus، تكون موسطات الدالة محدّدة بوضوح: إذ يمكن كتابة الدالة n^2 بالصيغة $\lambda n. n^2$ أو حتى بالصيغة $\lambda r. r^2$. إلاّ أنّ اعتماد طريقة تدوين أكثر دقة سيعقّد المعالجة الجبريّة، ولذلك فقد اخترنا أن نسمح بهذا التجاوز.

الداخلية محدود من الأعلى بـ $O(1)$ (أي ثابت)، وأعلى قيمة لكلٍّ من الدليلين i و j هي n على الأكثر، وتُنفَّذ الحلقة الداخلية مرّة واحدة على الأكثر لكل n^2 زوجًا من قيم i و j.

لمّا كان تدوين- O يصف حدًّا أعلى، فعندما نستخدمه لإعطاء حدٍّ لزمن تنفيذ خوارزمية ما في أسوأ الحالات، فسنحصل على حد لزمن تنفيذ الخوارزمية على أي مُدخل. وهكذا، فإن الحدّ $O(n^2)$ لزمن تنفيذ الفرز بالإدراج في أسوأ الحالات ينطبق أيضًا على زمن تنفيذ هذه الخوارزمية على أي مُدخَل. أمّا فيما يخص الحدّ $\Theta(n^2)$ لزمن تنفيذ الفرز بالإدراج في أسوأ الحالات، فإن ذلك لا يقتضي أن هناك حدًّا $\Theta(n^2)$ على زمن تنفيذ الفرز بالإدراج في أسوأ الحالات على أي مُدْخَل. فعلى سبيل المثال، رأينا في الفصل الثاني، أنه عندما يكون المُدخل مفروزًا سلفًا، فإن الفرز بالإدراج يُنفّذ في زمن $\Theta(n)$.

رياضيًّا، يمثّل قولنا إن زمن تنفيذ الفرز بالإدراج هو $O(n^2)$ تجاوزًا، وذلك لأن زمن التنفيذ الفعلي عندما n محدّدة يتغيّر بتغيّر المُدْخل ذي الحجم n. ولكن ما نعنيه عندما نقول "زمن التنفيذ هو $O(n^2)$ هو أن هناك دالة $f(n)$ تحقّق أنها $O(n^2)$ بحيث أنه أيًّا كانت قيمة n، وأيًّا كان المُدْخل ذو الحجم n الذي نختاره، فإن زمن التنفيذ لهذا المدخل محدودٌ من الأعلى بقيمة $f(n)$. وهذا ما يكافئ قولنا إن زمن التنفيذ في أسوأ الحالات هو $O(n^2)$.

تدوين-Ω

يقدّم تدوين-Ω ***حدًّا أدنى بالمقاربة asymptotic lower bound***، تمامًا مثلما يقدّم تدوين- O حدًّا أعلى بالمقاربة. فإذا كانت $g(n)$ دالةً معطاة، فإننا نعني بالعبارة $\Omega(g(n))$ [والتي تلفظ "big-omega of g of n"، أو أحيانًا "omega of g of n" فقط] مجموعةَ الدوال

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\} .$$

يبيّن الشكل 1.3(ت) الحدس الذي يرتكز عليه تدوين-Ω: لكل قيم n عند n_0 وإلى يمينها، تكون قيمة الدالة $f(n)$ مساوية لـ $cg(n)$ أو أعلى منها.

اعتمادًا على تعاريف التدوينات المقاربة التي رأيناها حتى الآن، من السهل أن نبرهن المبرهنة الهامة التالية (انظر التمرين 1.3-5).

مبرهنة 1.3

أيًّا كانت الدالتان $f(n)$ و $g(n)$، يكون لدينا $f(n) = \Theta(g(n))$ إذا وفقط إذا كانت $f(n) = O(g(n))$ و $f(n) = \Omega(g(n))$. ■

كمثال على تطبيق هذه المبرهنة، برهاننا بأن $an^2 + bn + c = \Theta(n^2)$ أيًّا كانت الثوابت a و b و c، حيث $a > 0$، الذي نستنتج منه مباشرة أن $an^2 + bn + c = \Omega(n^2)$ و $an^2 + bn + c = O(n^2)$. عمليًّا، بدلاً من استخدام المبرهنة 1.3 للحصول على حديّن أعلى وأدنى من حدود محكمة بالمقاربة - كما فعلنا في هذا المثال - فإننا نستخدمها عادةً للبرهان على حدود محكمة بالمقاربة بدءًا من حدود عليا ودنيا بالمقاربة.

عندما نقول إن *زمن تنفيذ* أية خوارزمية (دون تحديد إضافي) هو $\Omega(g(n))$، فإننا نعني أنه *مهما كان المدخل المحدد ذو الحجم n الذي نختاره لكل قيم n*، فإن زمن التنفيذ على هذا المدخل، عندما تكون n كبيرة كفاية، هو على الأقل مضاعف ثابت من $g(n)$. وهذا يكافئ قولنا إننا نعطي حدًّا أدنى لزمن تنفيذ خوارزمية في أحسن الحالات. فمثلاً، زمن تنفيذ الفرز بالإدراج في أحسن الحالات هو $\Omega(n)$، وهذا يقتضي أن زمن تنفيذ الفرز بالإدراج هو $\Omega(n)$.

إذن، ينتمي زمن تنفيذ الفرز بالإدراج إلى $\Omega(n)$ و $O(n^2)$، إذ إنه يقع في أي مكان بين دالة خطية ما لـ n ودالة تربيعية لـ n. إضافة إلى ذلك، فإن هذه الحدود تكون محكمة بالمقاربة قدر الإمكان: فمثلاً زمن تنفيذ الفرز بالإدراج ليس $\Omega(n^2)$، لأنه يوجد مُدْخَل ينفِّذ الفرز بالإدراج عليه في زمن $\Theta(n)$ (مثال ذلك، إذا كان المدخل مفروزًا سلفًا). ومع ذلك، فهذا لا يناقض قولنا إن زمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو $\Omega(n^2)$، وذلك لأنه يوجد مُدخَل يجعل الخوارزمية تستغرق زمنًا $\Omega(n^2)$.

التدوين المقارب في المعادلات والمتراجحات

رأينا سابقًا كيف يمكن استخدام التدوين المقارب داخل الصيغ الرياضية. مثلاً: عندما قدّمنا للتدوين- O، كتبنا: "$n = O(n^2)$". وقد نكتب أيضًا $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$. ولكن كيف نفسر مثل هذه الصيغ؟

عندما يكون التدوين المقارب وحيدًا (أي إنه لا يمثل جزءًا من صيغة أكبر منه) على الطرف الأيمن للمعادلة (أو للمتراجحة)، كما في $n = O(n^2)$، فقد عرّفنا سابقًا إشارة المساواة على أنها تعني الانتماء إلى المجموعات: $n \in O(n^2)$. ومع ذلك، إذا ظهر التدوين المقارب في صيغة ما، فإننا نفسره عمومًا، على أنه يحلّ محل دالة غير مسماة لا يهمنا كثيرًا أن نعطيها اسمًا؛ فمثلاً، تعني الصيغة $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ أن $2n^2 + 3n + 1 = 2n^2 + f(n)$، حيث $f(n)$ دالة ما من المجموعة $\Theta(n)$. وفي هذه الحالة نجعل $f(n) = 3n + 1$، وهي فعلاً من $\Theta(n)$.

يمكن أن يساعد استخدام التدوين المقارب بهذه الطريقة على حذف التفاصيل غير الضرورية في معادلة ما. مثلاً، عبّرنا في الفصل الثاني عن زمن تنفيذ الفرز بالدمج في أسوأ الحالات باستخدام العلاقة العودية

$$T(n) = 2T(n/2) + \Theta(n) .$$

فإذا كنا معنيّين فقط بالسلوك المقارب لـ $T(n)$، فلا داعي لتحديد كل الحدود من الدرجة الأصغر بدقة؛ ومن المفهوم أنها تدخل كلها في الدالة غير المسمّاة التي يشار إليها بالحدّ $\Theta(n)$.

نشير أيضًا إلى أننا نفهم أن عدد الدوال غير المسمّاة في عبارة ما يكون مساويًا لعدد المرات التي يظهر فيها التدوين المقارب. فمثلاً، في العبارة

$$\sum_{i=1}^{n} O(i) ,$$

يوجد فقط دالة غير معروفة وحيدة (هي دالة في i). وهكذا، فإن هذه العبارة تختلف عن العبارة: $O(1) + O(2) + \cdots + O(n)$ التي لا يمكن في الواقع تفسيرها تفسيرًا محددًا.

يظهر التدوين المقارب في بعض الأحيان في الطرف الأيسر من معادلة كما في

$$2n^2 + \Theta(n) = \Theta(n^2) .$$

نفسر مثل هذه المعادلات باستخدام القاعدة التالية: *مهما كانت طريقة اختيار الدوال غير المسمّاة إلى يسار إشارة مساواة، هناك طريقة لاختيار الدوال غير المسمّاة إلى يمين إشارة المساواة بحيث تكون المعادلة محققة.* إذن، يكون معنى المعادلة في مثالنا السابق هو أنه في حالة *أية* دالة $f(n) \in \Theta(n)$، توجد دالة ما $g(n) \in \Theta(n^2)$ بحيث يكون $2n^2 + f(n) = g(n)$ لكل قيم n. وبعبارة أخرى، يقدّم الطرف الأيمن للمعادلة مستوًى أقل تفصيلاً من الطرف الأيسر.

ويمكن سَلْسَلَة عدّة علاقات من هذا النوع معًا كما في

$$\begin{aligned} 2n^2 + 3n + 1 &= 2n^2 + \Theta(n) \\ &= \Theta(n^2) . \end{aligned}$$

ويمكن تفسير كل معادلة على حدة باستخدام القاعدة السابقة. فالمعادلة الأولى تعني أنه توجد دالة $f(n) \in \Theta(n)$ بحيث يكون $2n^2 + 3n + 1 = 2n^2 + f(n)$ لكل قيم n. وتعني المعادلة الثانية أنه في حالة أية دالة $g(n) \in \Theta(n)$ (كالدالة $f(n)$ التي ورد ذكرها الآن) توجد دالة $h(n) \in \Theta(n^2)$ بحيث يكون $2n^2 + g(n) = h(n)$ لكل قيم n. لاحظ أن هذا التفسير يؤدي إلى $2n^2 + 3n + 1 = \Theta(n^2)$، وهذا ما تبينه لنا ببساطة سلْسَلَة المعادلات.

تدوين$-o$

إن الحد الأعلى المقارب الذي يقدمه تدوين$-O$ قد يكون مُحْكمًا بالمقاربة، وقد لا يكون كذلك. فمثلاً الحد $2n^2 = O(n^2)$ مُحْكمٌ بالمقاربة، غير أن الحد $2n = O(n^2)$ ليس كذلك. نستخدم تدوين$-o$ لنعبّر عن حد أعلى غير محكم بالمقاربة. نعرّف $o(g(n))$ صوريًّا [وتلفظ "little-oh of g of n"] على أنها المجموعة:

$o(g(n)) = \{f(n)$: for any positive constant $c > 0$, there exists a constant

$0 \le f(n) < cg(n)$ for all $n \ge n_0\}$.

مثلاً $2n = o(n^2)$، فيما $2n^2 \neq o(n^2)$.

إن تعريفَي تدوين-O وتدوين-o متشابهان؛ والفرق الأساسي بينهما هو أنه عندما $f(n) = O(g(n))$، فإن المتراجحة $0 \le f(n) \le cg(n)$ تكون محققة في حالة ثابت ما $c > 0$، أما في حالة $f(n) = o(g(n))$، فإن المتراجحة $0 \le f(n) < cg(n)$ تكون محققة أيًّا كان الثابت $c > 0$. يمكننا ببساطة أن نقول إنه في التدوين-o تصبح الدالة $f(n)$ مهملة بالنسبة إلى $g(n)$ عندما تسعى n إلى اللانهاية؛ أي:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \ . \tag{1.3}$$

ويستخدم بعض المؤلفين هذه النهاية كتعريف للتدوين-o؛ ونحن في هذا الكتاب نشترط أن تكون الدوال غير المسمّاة موجبة بالمقاربة.

تدوين-ω

بالمثل، تُماثِل علاقةُ تدوينِ-ω بتدوينِ-Ω علاقةَ تدوينِ-o بتدوين-O. نستخدم تدوين-ω لنشير إلى حدٍّ أدنى غير محكم بالمقاربة. وإحدى طرائق تعريف ذلك هي:

$f(n) \in \omega(g(n))$ إذا وفقط إذا $g(n) \in o(f(n))$.

ومع ذلك، فإننا نعرّف $\omega(g(n))$ صوريًّا [وتلفظ "little-omega of g of n"] على أنها المجموعة:

$\omega(g(n)) = \{f(n)$: for any positive constant $c > 0$, there exists a constant

$n_0 > 0$ such that $0 \le cg(n) < f(n)$ for all $n \ge n_0\}$.

فمثلاً $n^2/2 = \omega(n)$، فيما $n^2/2 \neq \omega(n^2)$، وتقتضي العلاقة $f(n) = \omega(g(n))$ أن يكون

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \ ,$$

إذا كانت هذه النهاية موجودة فعلاً. أي إن قيم $f(n)$ تصبح كبيرة بلا حدود مقارنة بـ $g(n)$ عندما تسعى n إلى اللانهاية.

مقارنة الدوال

تنطبق العديد من الخصائص العلاقاتية للأعداد الحقيقيّة على المقارنات بالمقاربة. نفترض فيما يلي أن $f(n)$ و $g(n)$ موجبان بالمقاربة.

التعدي:

$f(n) = \Theta(g(n))$ و $g(n) = \Theta(h(n))$ يقتضيان $f(n) = \Theta(h(n))$،

$f(n) = O(g(n))$ و $g(n) = O(h(n))$ يقتضيان $f(n) = O(h(n))$،

$f(n) = \Omega(g(n))$ و $g(n) = \Omega(h(n))$ يقتضيان $f(n) = \Omega(h(n))$،

$f(n) = o(g(n))$ و $g(n) = o(h(n))$ يقتضيان $f(n) = o(h(n))$،

$f(n) = \omega(g(n))$ و $g(n) = \omega(h(n))$ يقتضيان $f(n) = \omega(h(n))$.

الانعكاسية:

$f(n) = \Theta(f(n))$،

$f(n) = O(f(n))$،

$f(n) = \Omega(f(n))$.

التناظرية:

$f(n) = \Theta(g(n))$ إذا وفقط إذا $g(n) = \Theta(f(n))$.

التناظرية المنقولة:

$f(n) = O(g(n))$ إذا وفقط إذا $g(n) = \Omega(f(n))$،

$f(n) = o(g(n))$ إذا وفقط إذا $g(n) = \omega(f(n))$.

ولمّا كانت هذه الخصائص محققة في التدوينات المقاربة، فيمكننا أن نبني مقابلةً بين المقارنة بالمقاربة بين دالتين f و g، والمقارنة بين عددين حقيقيّين a و b:

$f(n) = O(g(n))$ مثل $a \leq b$،

$f(n) = \Omega(g(n))$ مثل $a \geq b$،

$f(n) = \Theta(g(n))$ مثل $a = b$،

$f(n) = o(g(n))$ مثل $a < b$،

$f(n) = \omega(g(n))$ مثل $a > b$.

نقول إن الدالة $f(n)$ ***أصغر بالمقاربة asymptotically smaller*** من $g(n)$ إذا كان $f(n) = o(g(n))$، وأن $f(n)$ هي ***أكبر بالمقاربة asymptotically larger*** من $g(n)$ إذا كان $f(n) = \omega(g(n))$.

إلا أن هناك خاصيّة من خواص الأعداد الحقيقية لا تنطبق على التدوين المقارب وهي:

الفصل الثلاثي: في حالة أي عددين حقيقيين a و b، يجب أن تتحقق حكمًا إحدى الحالات الثلاث التالية: $a < b$ أو $a = b$ أو $a > b$.

ومع أنه يمكن إجراء مقارنة بين أيِّ عددين حقيقيّين؛ فلا يمكن إجراء مقارنة بالمقاربة بين أيِّ دالتين. أي إنه إذا كانت لدينا دالتان $f(n)$ و $g(n)$، فيمكن ألاّ يتحقق $f(n) = O(g(n))$ ولا يتحقق $f(n) = \Omega(g(n))$. فمثلاً لا يمكن مقارنة الدالتين n و $n^{1+\sin n}$ باستخدام التدوين المقارب، لأن قيمة الأس في $n^{1+\sin n}$ تتذبذب بين 0 و 2، وتأخذ كل القيم فيما بينهما.

تمارين

1-1.3

لتكن $f(n)$ و $g(n)$ دالتَيْن موجبتَيْن بالمقاربة. استخدم التعريف الأساسي لتدوين−Θ كي تبرهن على أن $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

2-1.3

بيّن أنه في حالة أي ثابتَيْن حقيقيّين a و b حيث $b > 0$، يكون

$$(n + a)^b = \Theta(n^b) \ . \qquad (2.3)$$

3-1.3

اشرح سبب كون العبارة "زمن تنفيذ الخوارزمية A هو على الأقل $O(n^2)$" لا معنى لها.

4-1.3

هل $2^{n+1} = O(2^n)$؟ وهل $2^{2n} = O(2^n)$؟

5-1.3

أثبت صحة المبرهنة 1.3.

6-1.3

برهن أن زمن تنفيذ خوارزمية ما هو $\Theta(g(n))$ إذا وفقط إذا كان زمن تنفيذها في أسوأ الحالات هو $O(g(n))$، وزمن تنفيذها في أحسن الحالات هو $\Omega(g(n))$.

7-1.3

برهن أن $o(g(n)) \cap \omega(g(n))$ هو المجموعة الخالية.

8-1.3

يمكن أن نعمّم التدوين الذي نعتمده ليشمل حالة موسطين n و m يمكن أن يسعيا إلى اللانهاية على نحو مستقل بمعدلين مختلفين. في حالة دالة معطاة $g(n,m)$، نعني بالتدوين $O(g(n,m))$ مجموعة الدوال

$$\begin{aligned} O(g(n,m)) = \{f(n,m) :\ & \text{there exist positive constants } c, n_0, \text{ and } m_0 \\ & \text{such that } 0 \le f(n,m) \le cg(n,m) \\ & \text{for all } n \ge n_0 \text{ or } m \ge m_0\} \ . \end{aligned}$$

أعطِ التعاريف المقابلة لـ $\Omega(g(n,m))$ و $\Theta(g(n,m))$.

2.3 تدوينات قياسية ودوال شائعة

يراجع هذا المقطع بعض الدوال الرياضية والتدوينات القياسية، ويدرس العلاقات فيما بينها. ويوضّح أيضًا استخدامات التدوينات المقاربة.

الاطّراد

نقول عن دالة $f(n)$ إنها ***متزايدة باطّراد monotonically increasing*** إذا كان $m \leq n$ يقتضي أن يكون $f(m) \leq f(n)$. وبالمثل، تكون الدالة ***متناقصة باطّراد monotonically decreasing*** إذا كان $m \leq n$ يقتضي $f(m) \geq f(n)$. وتكون الدالة $f(n)$ ***متزايدة تمامًا strictly increasing*** إذا كان $m < n$ يقتضي أن يكون $f(m) < f(n)$، ***ومتناقصةً تمامًا strictly decreasing*** إذا كان $m < n$ يقتضي $f(m) > f(n)$.

الأرضيات والسقوف

إذا كان x عددًا حقيقيًّا، فإننا نرمز إلى أكبر عدد صحيح أصغر أو يساوي x بالرمز $\lfloor x \rfloor$ (يُقرأ "أرضيّة x")، وإلى أصغر عدد صحيح أكبر أو يساوي x بالرمز $\lceil x \rceil$ (يُقرأ "سقف x"). ويكون:

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1 \; . \qquad (3.3)$$

أيًّا كان العدد الحقيقي.
ويكون:

$$\lceil n/2 \rceil + \lfloor n/2 \rfloor = n \, ,$$

أيًّا كان العدد الصحيح n،
ويكون:

$$\left\lceil \frac{\lceil x/a \rceil}{b} \right\rceil = \left\lceil \frac{x}{ab} \right\rceil \, , \qquad (4.3)$$

$$\left\lfloor \frac{\lfloor x/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{x}{ab} \right\rfloor \, , \qquad (5.3)$$

$$\left\lceil \frac{a}{b} \right\rceil \leq \frac{a + (b-1)}{b} \, , \qquad (6.3)$$

$$\left\lfloor \frac{a}{b} \right\rfloor \geq \frac{a - (b-1)}{b} \, . \qquad (7.3)$$

أيًّا كان العدد الحقيقي $x \geq 0$ والعددان الصحيحان $a, b > 0$.

إن دالة الأرضيّة $f(x) = \lfloor x \rfloor$ متزايدة باطّراد، كما هو الحال في دالة السقف $f(x) = \lceil x \rceil$.

الحساب بالمقاس

إذا كان a عددًا صحيحًا و n عددًا صحيحًا موجبًا، فإن $a \bmod n$ هي ***باقي قسمة*** (***remainder*** أو ***residue***) a/n، أي:

$$a \bmod n = a - n\lfloor a/n \rfloor \ . \tag{8.3}$$

وينتج عن ذلك:

$$0 \le a \bmod n < n \ . \tag{9.3}$$

من المناسب بعد إعطاء تعريف جيّد لمفهوم باقي قسمة عدد صحيح على آخر، أن يكون هناك تدوينًا خاصًا لبيان تساوي بواقي القسمة. فإذا كان $(a \bmod n) = (b \bmod n)$، فإننا نكتب $a \equiv b \pmod n$، ونقول إن a ***يكافئ*** (***equivalent to***) b بالمقاس n. وبعبارة أخرى: إن $a \equiv b \pmod n$ إذا تساوى باقيا قسمة كل من a و b على n. وهذا يكافئ قولنا إن $a \equiv b \pmod n$ إذا وفقط إذا كان n يقسم $b - a$. ونكتب $a \not\equiv b \pmod n$ إذا كانت a لا تكافئ b بالمقاس n.

كثيرات الحدود

ليكن d عددًا طبيعيًّا، إن ***كثير حدود في*** n ***من الدرجة*** d هو دالة $p(n)$ صيغتها:

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

حيث تمثل الثوابت $a_0, a_1, \ldots, a_d$ ***مُعاملات coefficients*** كثير الحدود، و $a_d \neq 0$. يكون كثير حدود موجبًا بالمقاربة إذا وفقط إذا كان $a_d > 0$. وإذا كان $p(n)$ كثيرَ حدود موجبًا بالمقاربة من الدرجة d، فإن $p(n) = \Theta(n^d)$. وإذا كان الثابت الحقيقي $a \ge 0$، فإن الدالة n^a تكون متزايدة باطّراد، وإذا كان الثابت الحقيقي $a \le 0$، فإن الدالة n^a تكون متناقصة باطّراد. ونقول عن دالة ما $f(n)$ إنها ***محدودة بكثير حدود polynomially bounded*** إذا كان $f(n) = O(n^k)$ حيث k ثابت ما.

الأسيّات

إذا كانت $a > 0$ و m و n أعددًا حقيقية، فتكون لدينا علاقات المساواة التالية:

$$\begin{aligned} a^0 &= 1 \ , \\ a^1 &= a \ , \\ a^{-1} &= 1/a \ , \\ (a^m)^n &= a^{mn} \ , \\ (a^m)^n &= (a^n)^m \ , \\ a^m a^n &= a^{m+n} \ . \end{aligned}$$

وأيًّا كان n و $a \geq 1$، فإن الدالة a^n متزايدة باطّراد في n. وحيث يكون ذلك مناسبًا، سنفترض أن $0^0 = 1$.

يمكن الربط بين معدلات نمو كثيرات الحدود والدوال الأسيّة بالحقيقة التالية: أيًّا كان الثابتان الحقيقيان a و b حيث $a > 1$، فإن:

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0 \; . \tag{10.3}$$

وهذا ما يسمح لنا باستنتاج أن

$$n^b = o(a^n) \; .$$

وهكذا، فإن أية دالة أسيّة ذات أساس أكبر تمامًا من 1 تتزايد بسرعة أكبر من أي كثير حدود.

باستخدام الرمز e للإشارة إلى العدد ...2.71828، الذي هو أساس دالة اللغاريتم الطبيعي، وأيًّا كان العدد الحقيقي x، يكون لدينا

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!} \; , \tag{11.3}$$

حيث ترمز "!" إلى دالة العاملي التي سنعرّفها لاحقًا في هذا المقطع. وأيًّا كان العدد الحقيقي x، تكون لدينا المتراجحة

$$e^x \geq 1 + x \; , \tag{12.3}$$

وتتحقق المساواة فقط عندما $x = 0$. أما إذا كان $|x| \leq 1$، فلدينا التقريب

$$1 + x \leq e^x \leq 1 + x + x^2 \; . \tag{13.3}$$

وعندما $x \to 0$، فإن تقريب e^x بـ $1 + x$ جيد كفاية:

$$e^x = 1 + x + \Theta(x^2) \; .$$

(استخدمنا التدوين المقارب في هذه المعادلة لوصف السلوك الحدّي عندما $x \to 0$ بدلاً من $x \to \infty$.) ويكون لدينا:

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x \; . \tag{14.3}$$

لجميع قيم x.

اللغاريتمات

سنستخدم التدوينات التالية:

$\lg n = \log_2 n$, (لغاريتم اثناني)

$$\ln n = \log_e n \,, \qquad \text{(لغاريتم طبيعي)}$$
$$\lg^k n = (\lg n)^k \,, \qquad \text{(رفع إلى أس)}$$
$$\lg\lg n = \lg(\lg n) \,. \qquad \text{(تركيب)}$$

سنعتمد فيما بعد اصطلاحًا تدوينيًّا هامًّا، وهو أن *الدوال اللغاريتمية تُطبَّق فقط على الحدّ التالي لها مباشرة في أية صيغة*، أي إن $\lg n + k$ تعني $(\lg n) + k$ وليس $\lg(n+k)$. إذا كان $b > 1$ ثابتًا، فإن الدالة $\log_b n$ تكون متزايدة تمامًا لجميع قيم $n > 0$.

إذا كانت $a > 0$ و $b > 0$ و $c > 0$ و n أعدادًا حقيقيّة، فإن:

$$a = b^{\log_b a} \,,$$
$$\log_c(ab) = \log_c a + \log_c b \,,$$
$$\log_b a^n = n \log_b a \,,$$
$$\log_b a = \frac{\log_c a}{\log_c b} \,, \tag{15.3}$$
$$\log_b(1/a) = -\log_b a \,,$$
$$\log_b a = \frac{1}{\log_a b} \,,$$
$$a^{\log_b c} = c^{\log_b a} \,, \tag{16.3}$$

وذلك عندما يكون أساس اللغاريتم في أيٍّ من المعادلات السابقة لا يساوي 1.

اعتمادًا على المعادلة (15.3)، نجد أن تغيير أساس اللغاريتم من ثابت إلى آخر يغيّر قيمة اللغاريتم بعامل ثابت فقط، ولهذا فإننا سنستخدم غالبًا الرمز "$\lg n$" عندما لا نهتم بالعوامل الثابتة، كما هي الحال في التدوين-O. ويرى المعلوماتيّون أن 2 هو الأساس الطبيعي الأنسب للغاريتمات، لأن العديد من الخوارزميات وبنى المعطيات تتضمن تفريق المسائل إلى جزأين.

هناك نشر بسيط للسلسلة $\ln(1+x)$ عندما $|x| < 1$ هو:

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots \,.$$

وإذا كانت $x > -1$، فلدينا أيضًا المتراجحتان التاليتان:

$$\frac{x}{1+x} \le \ln(1+x) \le x \,, \tag{17.3}$$

وهذه المساواة تتحقق إذا كانت $x = 0$ فقط.

نقول عن دالة $f(n)$ إنها محدودة ***بكثير لغاريتمي*** ***Polylogarithmically bounded*** إذا كان $f(n) = O(\lg^k n)$ حيث k ثابت ما. يمكن الربط بين نمو كثيرات الحدود وكثيرات اللغاريتمات بوضع $\lg n$ بدلاً من n، و 2^a بدلاً من a في المعادلة (10.3) فنجد:

$$\lim_{n\to\infty}\frac{\lg^b n}{(2^a)^{\lg n}}=\lim_{n\to\infty}\frac{\lg^b n}{n^a}=0\ .$$

ويمكن أن نستنتج من هذه النهاية أن:

$$\lg^b n = o(n^a)$$

أيًّا كان الثابت $a > 0$؛ أي إن أية دالة كثير حدود موجبة تنمو أسرع من أية دالة كثير لغاريتمي.

العامليات

يُعرَّف الرمز $n!$ [ويُقرأ "n عاملي"] للأعداد الطبيعية $n \ge 0$ كالتالي:

$$n! = \begin{cases}1 & \text{if } n=0\ ,\\ n\cdot(n-1)! & \text{if } n>0\ .\end{cases}$$

أي إن: $n! = 1\cdot 2\cdot 3\cdots n$.

هناك حدّ أعلى ضعيف لدالّة العاملي هو: $n! \le n^n$، إذ إن كل حدّ في جداء العاملي هو على الأكثر n.
ويعطى ***تقريب سترلنغ Stirling's approximation*** بالصيغة الآتية:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)\ , \tag{18.3}$$

حيث e هي أساس اللغاريتم الطبيعي. وهذا التقريب يعطينا حدًّا أعلى أكثر التصاقًا بالعاملي، إضافة إلى حدٍّ أدنى. ويمكننا برهان ما يلي: (انظر التمرين 2.3-3)

$$\begin{aligned} n! &= o(n^n)\ ,\\ n! &= \omega(2^n)\ ,\\ \lg(n!) &= \Theta(n\lg n)\ , \end{aligned} \tag{19.3}$$

حيث يسمح تقريب سترلنغ برهان العلاقة (19.3). هذا وتتحقق المعادلة التالية أيضًا أيًّا كانت $n \ge 1$:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n} \tag{20.3}$$

حيث

$$\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}\ . \tag{21.3}$$

التكرار الدالي

نستخدم التدوين $f^{(i)}(n)$ للتعبير عن تطبيق الدالة $f(n)$ تكراريًّا i مرة على قيمة بدئيّة لـ n. فإذا كانت $f(n)$ دالة معرّفة على الأعداد الحقيقيّة، وكان i عددًا طبيعيًّا، فإننا نعرّف هذا التدوين عَوْديًّا على النحو الآتي:

$$f^{(i)}(n) = \begin{cases} n & \text{if } i = 0 \, , \\ f(f^{(i-1)}(n)) & \text{if } i > 0 \, . \end{cases}$$

فمثلاً، إذا كان $f(n) = 2n$، فإن $f^{(i)}(n) = 2^i n$.

دالة اللغاريتم المكرّر

نستخدم التدوين $\lg^* n$ [ويُقرأ "لوغ نجمة لـ n"] لنشير إلى دالة اللغاريتم المكرّر المعرّفة كالتالي: لتكن $\lg^{(i)} n$ معرّفةً وفق التعريف السابق، مع الدالة $f(n) = \lg n$. ولمّا كان لغاريتم الأعداد السالبة غير معرّفٍ، فإن الدالة تكون معرّفةً إذا كان $\lg^{(i-1)} n > 0$ فقط. تأكّدْ أنك تُميِّز بين $\lg^{(i)} n$ (وهي دالة اللغاريتم مطبقة i مرة على التوالي، بدءًا من المحدّد n)، وبين $\lg^i n$ (وهو لغاريتم n مرفوعًا إلى القوة i). وبذلك نعرّف دالة اللغاريتم المكرّر كالتالي:

$$\lg^* n = \min\{i \geq 0 : \lg^{(i)} n \leq 1\}$$

إن اللغاريتم المكرّر دالّة تتزايد ببطءٍ شديد:

$$\begin{aligned} \lg^* 2 &= 1 \, , \\ \lg^* 4 &= 2 \, , \\ \lg^* 16 &= 3 \, , \\ \lg^* 65536 &= 4 \, , \\ \lg^*(2^{65536}) &= 5 \, . \end{aligned}$$

ولمّا كان عدد الذرات في العالَم الممكن ملاحظته يُقدَّر بنحو 10^{80}، وهو أقلّ بكثير من 2^{65536}، فمن النادر أن نقع على مُدخَل حجمه n بحيث يكون $\lg^* n > 5$.

أعداد فيبوناتشي

تعرَّف ***أعداد فيبوناتشي*** ***Fibonacci numbers*** بالعلاقة العَوْدية التالية:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2} \quad \text{for } i \geq 2 \, . \end{aligned} \tag{22.3}$$

وهكذا، فإن أي عدد من أعداد فيبوناتشي هو مجموع العددين اللذين يسبقانه، وهذا ما يعطي المتتالية

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$.

إن أعداد فيبوناتشي مرتبطة ***بالكسر الذهبيّ*** ***golden ratio*** ϕ، وبمرافقه $\hat{\phi}$، وهما جذرا المعادلة

$$x^2 = x + 1 \tag{23.3}$$

وهما معرّفان بالصيغتين التاليتين (انظر التمرين 3.2-6):

$$\phi = \frac{1+\sqrt{5}}{2}$$
$$= 1.61803\ldots ,$$
$$\hat{\phi} = \frac{1-\sqrt{5}}{2}$$
$$= -.61803\ldots .$$

وتحديدًا لدينا

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}} ,$$

وهذا ما يمكن برهانه بالاستقراء (التمرين 2.3-7). ولما كان $|\hat{\phi}| < 1$، فلدينا

$$\frac{|\hat{\phi}^i|}{\sqrt{5}} < \frac{1}{\sqrt{5}}$$
$$< \frac{1}{2} ,$$

وهذا يقتضي أن

$$F_i = \left\lfloor \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} \right\rfloor , \tag{25.3}$$

أي إن عدد فيبوناتشي F_i ذي الرقم i، يساوي $\phi^i/\sqrt{5}$ مدوّرًا إلى أقرب عدد طبيعي. إذن، فإن أعداد فيبوناتشي تتزايد أسيًّا.

تمارين

1-2.3

برهن أنه إذا كانت $f(n)$ و $g(n)$ دالتين متزايدتين باطّراد، فإن $f(n)+g(n)$ و $f(g(n))$، متزايدتان باطراد. وإذا كانت $f(n)$ و $g(n)$ دالتين متزايدتين باطّراد وموجبتين، فإن $f(n)\cdot g(n)$ متزايدة باطّراد أيضًا.

2-2.3

برهن المعادلة (16.3).

3-2.3

برهن المعادلة (19.3). وبرهن أيضًا أن $n! = \omega(2^n)$، وأن $n! = o(n^n)$.

4-2.3 ⋆

هل الدالة $\lceil \lg n \rceil!$ محدودة بكثير حدود؟ وهل الدالة $\lceil \lg\lg n \rceil!$ محدودة بكثير حدود؟

5-2.3 ⋆

أيُّ دالة أكبر بالمقارنة: $\lg(\lg^* n)$ أم $\lg^*(\lg n)$؟

6-2.3

بيّن أن الكسر الذهبي ϕ ومرافقه $\widehat{\phi}$ يحققان معًا المعادلة $x^2 = x + 1$.

7-2.3

برهن بالاستقراء أن عدد فيبوناتشي ذا الرقم i يحقق المساواة:

$$F_i = \frac{\phi^i - \widehat{\phi}^i}{\sqrt{5}} ,$$

حيث ϕ هو الكسر الذهبي و $\widehat{\phi}$ مرافقه.

8-2.3

بيّن أن $k \ln k = \Theta(n)$ يقتضي $k = \Theta(n/\ln n)$.

مسائل

1-3 السلوك المقارب لكثيرات الحدود

ليكن

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

حيث $a_d > 0$، كثير حدود في n من الدرجة d، وليكن k ثابتًا ما. استخدم تعاريف التدوينات المقاربة للبرهان على الخصائص التالية:

أ. إذا كان $k \geq d$، فإن $p(n) = O(n^k)$.

ب. إذا كان $k \leq d$، فإن $p(n) = \Omega(n^k)$.

ت. إذا كان $k = d$، فإن $p(n) = \Theta(n^k)$.

ث. إذا كان $k > d$، فإن $p(n) = o(n^k)$.

ج. إذا كان $k < d$، فإن $p(n) = \omega(n^k)$.

2-3 النمو المقارب النسبي

في الجدول الآتي لدينا أزواج العبارات (A,B)، بيّن هل A هو O أو o أو Ω أو ω أو Θ لـ B؟ افترض أن $k \geq 1$ و $\epsilon > 0$ و $c > 1$ ثوابت. يجب أن يكون جوابك بصيغة "نعم" أو "لا" في كل خانة من خانات الجدول.

	A	B	O	o	Ω	ω	Θ
أ.	$\lg^k n$	n^ϵ					
ب.	n^k	c^n					
ت.	$\sqrt{n}$	$n^{\sin n}$					
ث.	2^n	$2^{n/2}$					
ج.	$n^{\lg c}$	$c^{\lg n}$					
ح.	$\lg(n!)$	$\lg(n^n)$					

3-3 *الترتيب وفق معدلات النمو المقاربة*

أ. رتِّب الدوال التالية وفق ترتيب نموّها؛ أي أوجد الترتيب $g_1, g_2, \ldots, g_{30}$ الذي يحقق: $g_1 = \Omega(g_2),\ g_2 = \Omega(g_3), \ldots, g_{29} = \Omega(g_{30})$. قسّم قائمتك إلى صفوف تكافؤ بحيث تكون الدالتان $f(n)$ و $g(n)$ في الصف نفسه إذا وفقط إذا كان $f(n) = \Theta(g(n))$.

$\lg(\lg^* n)$	$2^{\lg^* n}$	$(\sqrt{2})^{\lg n}$	n^2	$n!$	$(\lg n)!$
$\left(\frac{3}{2}\right)^n$	n^3	$\lg^2 n$	$(\lg n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$\lg^* n$	$n \cdot 2^n$	$2^{\lg \lg n}$	$\ln n$	1
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$
$\lg^*(\lg n)$	$2^{\sqrt{2 \lg n}}$	n	2^n	$n \lg n$	$2^{2^{n+1}}$

ب. أعطِ مثالاً لدالة موجبة واحدة $f(n)$ بحيث لا تحقق $O(g_i(n))$ ولا $\Omega(g_i(n))$، مع أيٍّ من الدوال $g_i(n)$ الواردة في الجزء السابق (أ).

4-3 *خواص التدوين المقارب*

لتكن $f(n)$ و $g(n)$ دالتين موجبتين بالمقاربة. برهن صحة أو عدم صحة كلٍّ من المخمَّنات التالية:

أ. $f(n) = O(g(n))$ يقتضي $g(n) = O(f(n))$.

ب. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

ت. $f(n) = O(g(n))$ يقتضي $\lg(f(n)) = O(\lg(g(n)))$، حيث $\lg(g(n)) \geq 1$ و $f(n) \geq 1$ لكل قيم n الكبيرة كفاية.

ث. $f(n) = O((g(n))$ يقتضي $2^{f(n)} = O(2^{g(n)})$.

ج. $f(n) = O((f(n))^2)$.

ح. $f(n) = O(g(n))$ يقتضي $g(n) = \Omega(f(n))$.

خ. $f(n) = \Theta(f(n/2))$.

د. $f(n) + o(f(n)) = \Theta(f(n))$.

5-3 أشكال معدّلة من O و Ω

يُعرِّف بعض المؤلّفين Ω بطريقة مختلفة قليلاً عن الطريقة التي عرّفناها بها هنا. نستخدم لهذا التعريف البديل الرمزَ $\overset{\infty}{\Omega}$ (ويقرأ "أوميغا لانهاية")، ونقول إن $f(n) = \overset{\infty}{\Omega}(g(n))$ إذا وُجد ثابت موجب c بحيث يكون $0 \leq cg(n) \leq f(n)$ لعددٍ لانهائيٍّ من القيم الصحيحة n.

أ. بيِّن أنه أيًّا كانت الدالتان $f(n)$ و $g(n)$ الموجبتان بالمقاربة، فإما أن يكون $f(n) = O((g(n))$، وإما أن يكون $f(n) = \overset{\infty}{\Omega}(g(n))$، وإما يكونا معًا، على حين أن هذا لا يكون صحيحًا إذا وضعنا Ω مكان $\overset{\infty}{\Omega}$.

ب. اشرح المحاسن والمساوئ الكامنة في استخدام $\overset{\infty}{\Omega}$ مكان Ω في توصيف أزمنة تنفيذ البرامج.

يُعرِّف بعض المؤلّفين O أيضًا بطريقة مختلفة قليلاً. نستخدم لهذا التعريف البديل الرمزَ O'، ونقول إن $f(n) = O'((g(n))$ إذا وفقط إذا $|f(n)| = O(g(n))$

ت. ما الذي يطرأ على اتجاهَي الاقتضاء "إذا وفقط إذا" في المبرهنة 1.3 إذا وضعنا O' مكان O، وحافظنا على Ω؟

يستعمل بعض المؤلّفين الرمز $\tilde{O}$ (ويُقرأ "Soft-oh")، للدلالة على O مع إهمال العوامل اللغاريتميّة، ويعرِّفونه كما يلي:

$\tilde{O}(g(n)) = \{f(n)$: there exist positive constants $c, k,$ and n_0 such that

$$0 \leq f(n) \leq cg(n) \lg^k(n) \text{ for all } n \geq n_0\} .$$

ث. عرِّف بأسلوب مماثل كلاًّ من: $\tilde{\Omega}$ و $\tilde{\Theta}$، ثم برهن المبرهنة 1.3 باستخدام هذه التدوينات المعدَّلة.

3-6 الدوال المكررة

يمكن تطبيق عملية التكرار * المستخدمة في دالة $\lg^*$ على أية دالة متزايدة بانتظام $f(n)$ معرّفة على الأعداد الحقيقية. فإذا كان الثابت $c \in \mathbb{R}$، فإننا نعرّف الدالة المكرّرة f_c^* على أنها

$$f_c^*(n) = \min\{i \geq 0 : f^{(i)}(n) \leq c\} ,$$

والتي ليس من الضروري أن تكون معرّفة في كل الحالات. وبعبارة أخرى: إن قيمة $f_c^*(n)$ هي عدد مرات التطبيق المتتالي للدالة f اللازم لتتناقص قيمتها حتى c أو أقل.

أعطِ لكلٍّ من الدوال $f(n)$ التالية والثوابت c، حدًّا محكمًا قدر الإمكان على $f_c^*(n)$.

	$f(n)$	c	$f_c^*(n)$
أ.	$n-1$	0	
ب.	$\lg n$	1	
ت.	$n/2$	1	
ث.	$n/2$	2	
ج.	$\sqrt{n}$	2	
ح.	$\sqrt{n}$	1	
خ.	$n^{1/3}$	2	
د.	$n/\lg n$	2	

ملاحظات الفصل

يعيد Knuth [209] أصل تدوين$-O$ إلى نصٍّ في نظرية الأعداد بقلم P.Bachmann يعود إلى العام 1892. وقد ابتدع E.Landau تدوين$-o$ في عام 1909 ليستخدمه في مناقشته توزيع الأعداد الأوليّة. وقد ابتدع Knuth [213] تدوينَي Ω و Θ ليصحّح الاستخدام الشائع في الأدبيات، رغم أنه غير دقيق تقنيًّا، لتدوين$-O$ لحدودٍ عليا ودنيا على السواء. وما زال الكثير يستخدمون تدوين$-O$ حيث إن تدوين$-\Theta$ هو أكثر دقّة تقنيًّا. المزيد من المناقشة حول تاريخ وتطوّر التدوينات المقاربة موجود في Knuth [209, 213] وفي Brassard و Bratley [55].

لا يُجمِع المؤلفون على تعريف التدوينات المقاربة بالطريقة نفسها، إلا أن التعاريف المختلفة تتفق في معظم الحالات الشائعة. تشمل بعض التعاريف دوال غير موجبة بالمقاربة، مادامت قيمها المطلقة محدودة كما ينبغي.

تُنسب المعادلة (20.3) إلى Robbins [297]. ويمكن العثور على خواص أخرى للدوال الرياضيّة الأساسيّة في أي مرجع جيّد في الرياضيات، مثل Abramowitz و Stegun [1] أو Zwillinger [362]، أو في كتابٍ في حساب التفاضل والتكامل مثل Apostol [18] أو Thomas وآخرون [334]. ويضمّ كلٌّ من Knuth [209] و Graham و Knuth و Patashnik [152] مادةً غنيةً في الرياضيات المتقطعة المستخدمة في علوم الحاسوب.

4 فرّق–تسد

رأينا في المقطع 1.3.2 أن الفرز بالدمج يقدم مثالاً عن نموذج خوارزميات فرّق–تسد. تذكّر أننا في سياق فرّق–تسد، نحلّ مسألة ما عَوْديًا، بتطبيق ثلاث خطوات في كل مستوًى من العَوْدية:

فرِّقْ المسألةَ إلى عدد من المسائل الجزئية التي هي منتسخات أصغر من المسألة نفسها.

سُدْ أي سيطر على المسائل الجزئية بحلّها عَوْديًا. في حال كانت حجوم المسائل الجزئية صغيرة كفاية، يكفيك أن تحلّها بطريقة مباشرة.

جَمِّعْ حلول المسائل الجزئية في حل للمسألة الأصلية.

عندما تكون حجوم المسائل الجزئية كبيرة كفاية لحلها عَوْديًا، فإننا ندعو ذلك ***بالحالة العَوْدية*** ***recursive case***. ما إن تصبح المسائل الجزئية صغيرة كفاية بحيث نتوقف عن تقسيمها عَوْديًا، نقول إن العَوْدية قد وصلت إلى "القاع"، وإننا قد نزلنا إلى ***الحالة الأساسية*** ***base case***. قد نضطر، في بعض الأحيان – إضافة إلى حل المسائل الجزئية التي هي منتسخات أصغر من المسألة نفسها – إلى حل مسائل جزئية لا تشبه تمامًا المسألة الأصلية. نعُدُّ حل مثل هذه المسائل الجزئية جزءًا من خطوة التجميع.

سنطّلع في هذا الفصل على المزيد من الخوارزميات المعتمدة على طريقة فرّق–تسد. الخوارزمية الأولى تحلّ مسألة الصفيفة الجزئية العظمى: وفيها يكون دخلُ الخوارزمية صفيفةً من الأعداد، وتحدِّد الصفيفة الجزئية المتتاليةَ ذات المجموع الأكبر. ثم نطّلع على خوارزميتي فرّق–تسد لحساب جداء مصفوفات $n \times n$. تُنفّذ الأولى في زمن $\Theta(n^3)$، أي إنها لا تتفوق على الطريقة المباشرة في جداء المصفوفات المربعة، فيما تُنفّذ الثانية، خوارزمية شتراسن Strassen، في زمن $\Theta(n^{2.81})$، وهذا الزمن يتفوق على زمن الطريقة المباشرة بالمقاربة.

العلاقات العَوْدية

تقترن العلاقات العَوْدية بطريقة فرّق–تسد لأنها تعطينا طريقة طبيعيّة لتوصيف أزمنة تنفيذ الخوارزميات التي تتبع هذه الطريقة. ***العلاقة العَوْدية*** ***recurrence*** هي معادلة أو متراجحة توصِّف دالةً بدلالة قيمها نفسها لمُدْخلات أصغر منها. رأينا مثلاً، في المقطع 2.3.2، أنه يمكن توصيف $T(n)$ زمن تنفيذ إجرائية

MERGE-SORT في أسوأ الحالات باستخدام العلاقة العَوْدية

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 , \end{cases} \tag{1.4}$$

وذكرنا أن حلّها هو $T(n) = \Theta(n \lg n)$.

يمكن أن تأخذ العلاقات العَوْدية صيغًا عديدة. فعلى سبيل المثال قد تقسّم خوارزميةٌ عَوْديةٌ المسألةَ إلى مسائل جزئية من أحجام غير متساوية، كأن نفرقها مثلاً 2/3 إلى 1/3. إذا كانت خطوتا التقسيم والتجميع تستغرقان زمنًا خطيًّا، فيستنتج عن هذه الخوارزمية العلاقة العَوْدية $T(n) = (2n/3) + T(n/3) + \Theta(n)$.

ليس من الضروري أن تكون المسائل الجزئية ذات أجزاء ثابتة من حجم المسألة الأصلية. فمثلاً، قد تقوم نسخة عَوْديّة من البحث الخطي (انظر التمرين 2.1-3) بتوليد مسألة جزئية تحتوي على عدد من العناصر يقل عن المسألة الأصلية بواحد فقط. وسيستغرق كل استدعاء عَوْدي زمنًا ثابتًا إضافة إلى زمن الاستدعاءات العَوْدية التي يقوم بها، وهذا يعطي العلاقة العَوْدية $T(n) = T(n-1) + \Theta(1)$.

يقدّم هذا الفصل ثلاث طرق لحل العلاقات العَوْدية- أي للحصول على حدودٍ مقاربةٍ لحلّها معبَّرٍ عنها بـ "O" أو "Θ":

- في ***طريقة التعويض substitution method***، نخمن حدًّا ونستخدم الاستقراء الرياضي لنبرهن على صحّة تخميننا.
- تحوّل طريقة ***شجرة العَوْدية recursion-tree*** العلاقةَ العَوْدية إلى شجرة، تمثل عقدها التكاليف الواجبة في مختلف مستويات العَوْدية، ثم نستخدم تقنيات لحدّ سلاسل الجمع في حل العلاقة العَوْدية.
- تقدم ***الطريقة الرئيسة master method*** حدودًا للعلاقات العَوْدية من الصيغة

$$T(n) = aT(n/b) + f(n) , \tag{2.4}$$

حيث $a \geq 1$، و $b > 1$، و $f(n)$ دالة معطاة. تصادفنا مثل هذه العلاقات العَوْدية مرارًا. فعلاقة عَوْدية مثل تلك في المعادلة (2.4) توصّف خوارزمية فرّق-تسد التي تنشئ a مسألة جزئية، حجم كلٍّ منها $1/b$ من حجم المسألة الأصلية، وتستغرق فيها خطوتا التقسيم والتجميع معًا زمنًا $f(n)$.

ويتطلب استخدام الطريقة الرئيسة أن تحفظ ثلاث حالات في ذاكرتك، ولكن ما إن تتمكن من حفظها، حتى يصبح تحديد حدود مقاربة العديد من العلاقات العَوْدية البسيطة سهلاً عليك. سنستخدم الطريقة الرئيسة في تحديد أزمنة تنفيذ خوارزميات فرّق-تسد لمسألة الصفيفة الجزئية العظمى ولجداء المصفوفات، ولخوارزميات أخرى موجودة في هذا الكتاب تعتمد على مبدأ فرّق-تسد.

ستصادفنا أحيانًا علاقات عَوْدية بصيغة متراجحات لا بصيغة علاقات مساواة، مثل $T(n) \leq 2T(n/2) + \Theta(n)$. ولما كانت مثل هذه العلاقات تعطي حدًّا أعلى لـ $T(n)$ فقط، فإننا

سنصوغ حلها باستخدام تدوين-O بدلاً من تدوين-Θ. وبالمثل، إذا عُكست المتراجحة لتصبح $T(n) \geq 2T(n/2) + \Theta(n)$، فنستخدم تدوين-Ω في حلّها لأنها أصلاً تعطي حدًّا أدنى لـ $T(n)$ فقط.

تفاصيل تقنيّة في العلاقات العَوْدية

عمليًّا، عندما نستعرض علاقات عَوْديّة ونحلّها، فإننا نهمل بعض التفاصيل التقنية. على سبيل المثال، إذا استدعينا MERGE-SORT لفرز n عنصرًا، عندما يكون n فرديًّا، فإننا نحصل على مسألتين حجمهما $\lfloor n/2 \rfloor$ و $\lceil n/2 \rceil$. وأيٌّ من هاتين القيمتين لا تساوي فعلاً $n/2$، لأن $n/2$ ليس عددًا طبيعيًّا عندما يكون n فرديًّا. إن العلاقة التي تصف رياضيًّا زمن تنفيذ إجرائية MERGE-SORT في أسوأ الحالات هو فعليًّا:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\ , \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1\ . \end{cases} \tag{3.4}$$

وتمثل الشروط الحديّة نمطًا آخر من التفاصيل التي نتجاهلها عادة. ولما كان زمنُ تنفيذِ خوارزميةٍ ما – دخلٍ ذي حجمٍ ثابتٍ – ثابتًا أيضًا، فإن صيغة العلاقات العَوْدية الناتجة عن أزمنة تنفيذ الخوارزميات هي $T(n) = \Theta(1)$ عمومًا، حيث n صغيرة كفاية. ولهذا، ولغرض التبسيط، سنهمل عمومًا عبارات الشروط الحديّة للعلاقات العَوْدية، وسنفترض أن $T(n)$ ثابتة عندما تكون n صغيرة. فعلى سبيل المثال، نكتب عادة العلاقة العَوْدية في (1.4) بالصيغة

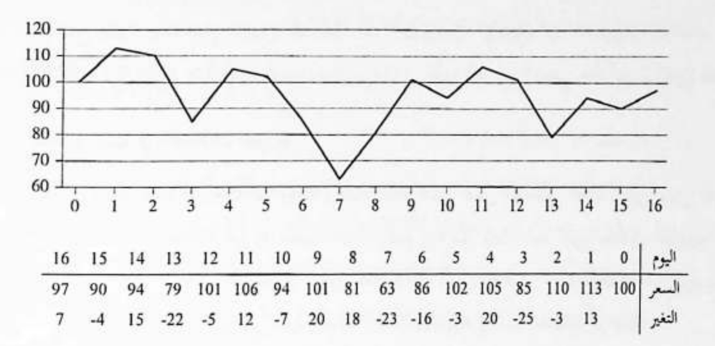
$$T(n) = 2T(n/2) + \Theta(n)\ , \tag{4.4}$$

دون أن نوضّح صراحة القيم عندما تكون n صغيرة. والسبب هو أنه على الرغم من تغيّر حل العلاقة العَوْديّة بتغيّر قيمة $T(1)$، فإن الحل لا يتغيّر عادة إلاّ بعامل ثابت، وهكذا تبقى مرتبة النمو نفسها دون تغيير.

عندما نعطي العلاقات العَوْديّة ونحلّها، فغالبًا ما نحذف الأرضيات والأسقف والشروط الحديّة. إننا نتابع قُدُمًا دون هذه التفاصيل، ثمّ نقرّر لاحقًا: هل هي مهمة أم لا؟ ومع أنها غالبًا ما تكون غير مهمة، فينبغي معرفة متى تكون مهمة فعلاً. وتساعد على ذلك الخبرة، إضافةً إلى بعض المبرهنات التي تشير إلى أن هذه التفاصيل لا تؤثر في الحدود المقاربة لكثيرٍ من العلاقات العَوْديّة التي تصادفنا عند تحليل خوارزميات فرّق-تسد (انظر المبرهنة 1.4). على أية حال، سنعالج في هذا الفصل بعض هذه التفاصيل ونبيّن النقاط الدقيقة المتعلقة بطرق حل العلاقات العَوْديّة.

1.4 مسألة الصفيفة الجزئية العظمى

افترض أنه أتيحت لك الفرصة أن تستثمر في شركة Volatile Chemical Corporation. وكما هي حال المواد الكيمياوية الطيّارة التي تنتجها الشركة، فإن سعر سهم الشركة طيّار أيضًا ومتذبذب. يُسمَح لكَ أن تشتري وحدة من الأسهم مرّة واحدة فقط لتبيعها في موعد لاحق، على أن يتمّ البيع والشراء بعد إغلاق



اليوم	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
السعر	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
التغير		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

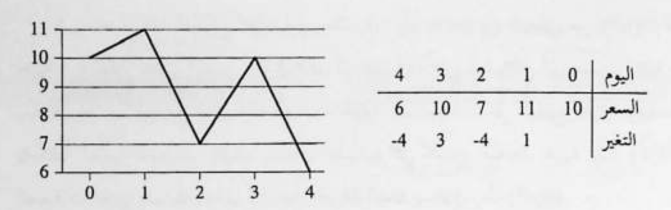
الشكل 1.4 معلومات عن سعر سهم شركة Volatile Chemical Corporation بعد إغلاق المبادلات خلال مدة 17 يومًا. يبيّن المحور الأفقي للمخطط البياني اليوم، ويبيّن المحور العمودي السعر. ويبين السطر الأسفل في الجدول تحت المخطط تغيّر السعر مقارنة باليوم السابق.

المداولات في نهاية اليوم. للتعويض عن هذا القيد، يُسمَحُ لك أن تعرف سعر السهم في المستقبل. هدفك هو الحصول على الربح الأعظم. يبيّن الشكل 1.4 سعر السهم خلال مدة طولها 17 يومًا. يمكنك الشراء، في أي وقت تريده ولمرّة واحدة، بعد اليوم 0 الذي يكون فيه سعر السهم $100. سترغب طبعًا في أن "تشتري بسعر منخفض، وتبيع بسعر مرتفع" - بالأحرى أن تشتري بأخفض سعر ممكن، ثمّ تبيع بعد ذلك بأعلى سعر ممكن – وذلك لتعظِّم ربحك. ولكن لسوء الحظ، قد لا يكون بإمكانك فعلاً الشراء بأدنى سعر ممكن ثم البيع بأعلى سعر ممكن خلال مدّة معطاة. ففي الشكل 1.4 نرى أن أدنى سعر يتحقق بعد اليوم السابع، وهو يلي اليوم الأول الذي يصل فيه السعر إلى أعلى قيمة.

قد تفكر في أنه بإمكانك أن تعظِّم الربح سواءً بالشراء دائمًا بأخفض سعر أو بالبيع دائمًا بأعلى سعر. على سبيل المثال، قد نعظِّم الربح بالشراء بالسعر الأدنى، بعد اليوم 7. لو نجحت هذه الاستراتيجية دائمًا، لكان من السهل تحديد طريقة لتعظيم الربح: أوجد أعلى سعر وأدنى سعر، ثمّ عُدْ نحو اليسار بدءًا من أعلى سعر لتحدّد أدنى سعر سابق له، أو ابدأ من أدنى سعر وتقدمْ يمينًا لتجد أعلى سعر لاحق له، ثمّ اختر زوج الأسعار الذي يحقق أكبر فارق. يبيّن الشكل 2.4 مثالاً معاكسًا بسيطًا يبيّن أن الربح الأعظم أحيانًا لا يأتي بالشراء عند أدنى سعر ولا بالبيع عند أعلى سعر.

حل فجّ

يمكننا بسهولة إعطاء حلٍّ فجٍّ لهذه المسألة: يكفي أن نجرِّب كل زوجين من أيام شراء ومبيع يكون فيه يوم الشراء سابقًا ليوم البيع. يمكن في مدّة من n يومًا إيجاد $\binom{n}{2}$ زوجًا من هذه الأيام. ولما كان $\binom{n}{2}$ من رتبة



اليوم	0	1	2	3	4
السعر	10	11	7	10	6
التغير		1	-4	3	-4

الشكل 2.4 مثال يبيّن أن الربح الأعظم لا يتحقق دائمًا بالبدء بالسعر الأدنى أو بالانتهاء بالسعر الأعلى. مرّة ثانية، يشير المحور الأفقي إلى اليوم، ويبين المحور العمودي السعر. الربح الأعظم هو $3 للسهم الواحد، ويمكن تحقيقه بالشراء بعد اليوم 2 وبالبيع بعد اليوم 3. ليس السعر $7 بعد اليوم 2 هو السعر الأدنى عمومًا ، ولا السعر $10 بعد اليوم 3 هو السعر الأعلى عمومًا.

$\Theta(n^2)$، وكان أفضل ما يمكن أن نأمله هو أن يستغرق حساب كلّ زوج من الأيام زمنًا ثابتًا، فإن مثل هذه الطريقة ستستغرق زمنًا $\Omega(n^2)$. هل بإمكاننا تحسين ذلك؟

تحويل

بهدف تصميم خوارزمية بزمن تنفيذ $o(n^2)$، سننظر إلى الدخل بطريقة مختلفة قليلاً. نريد أن نجد متتالية من الأيام يكون الفارق الصافي من أول يوم وحتى آخر يوم أعظميًّا. فبدلاً من النظر إلى السعر اليومي لننظر إلى التغيّر اليومي في السعر، حيث يكون التغيّر في اليوم i هو الفارق بين سعر نهاية اليوم $i-1$ وسعر نهاية اليوم i. يبيّن الجدول في الشكل 1.4 هذه التغيّرات في السطر الأسفل. إذا كنا نتعامل مع هذا السطر على أنه صفيفة A كتلك المبينة في الشكل 3.4، فإن هدفنا الآن هو إيجاد صفيفة جزئية متتالية وغير خالية من A يكون مجموع عناصرها هو الأكبر. نسمّى هذه الصفيفة الجزئية المتتالية ***الصفيفة الجزئية العظمى*** ***maximum subarray***. على سبيل المثال: في الصفيفة المبينة في الشكل 3.4، الصفيفة الجزئية العظمى من $A[1..16]$ هي $A[8..11]$، مع المجموع 43. إذن، سنرغب في الشراء مباشرة قبل اليوم 8 (أي في نهاية اليوم 7) وأن نبيع بعد نهاية اليوم 11، لنحقق بذلك ربحًا يصل إلى $43 للسهم الواحد.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

صفيفة جزئية عظمى

الشكل 3.4 التغيّر في أسعار الأسهم باعتباره مسألة صفيفة جزئية عظمى. في هذا المثال، تحقّق الصفيفة الجزئية $A[8..11]$ ذات المجموع 43، المجموع الأعلى من بين أية صفيفة جزئية ملامسة للصفيفة A.

لا يساعدنا هذا التحويل كثيرًا، أول وهلة. فما زلنا بحاجة إلى التحقق من $\binom{n-1}{2} = \Theta(n^2)$ صفيفة جزئية لـ n يومًا. يطلب التمرين 4.1-2 إليك أن تبيّنَ أنه، على الرغم من أن حساب كلفة صفيفة جزئية قد يستغرق زمنًا متناسبًا مع طول هذه الصفيفة الجزئية عند حساب كل المجاميع الجزئية وعددها $\Theta(n^2)$، فإن بإمكاننا تنظيم الحسابات بحيث يستغرق حساب كل مجموع صفيفة جزئية زمنًا $O(1)$، بمعرفة مجاميع الصفيفات الجزئية السابقة. وعليه فإن حل الطريقة الفجّة يستغرق زمنًا $\Theta(n^2)$.

فلنبحث إذن عن حل أكثر فعالية لمسألة الصفيفة الجزئية العظمى. وعندما نقوم بذلك، فإننا نتحدث عادة عن "صفيفةٍ جزئيةٍ عظمى" وليس عن "الصفيفة الجزئية العظمى"، إذ قد يكون هناك أكثر من صفيفة جزئية تحقّق المجموع الأعظم.

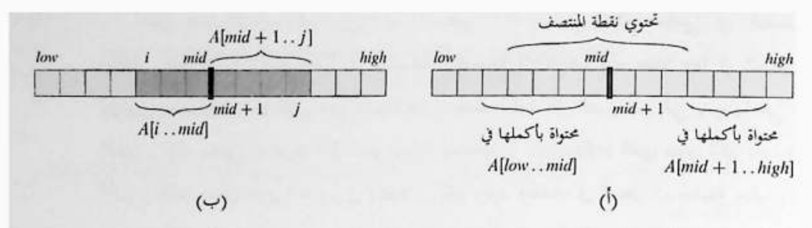
إن مسألة الصفيفة الجزئية العظمى تكون مثيرة للاهتمام فقط عندما تحتوي الصفيفة بعض الأعداد السالبة. فعندما تكون كل مركبات الصفيفة موجبةً، فإن حل المسألة لن يكون فيه أي تحدٍّ، إذ إن كامل الصفيفة يحقق المجموع الأعلى.

حل باستخدام فرّق-تسد

دعنا نفكر كيف يمكن أن نحلّ مسألة الصفيفة الجزئية العظمى باستخدام تقنيّة فرّق-تسد. افترض أننا نريد أن نجد صفيفة جزئية عظمى من الصفيفة الجزئية $A[low..high]$. تقترح طريقة فرّق-تسد أن تقسّم الصفيفة الجزئية إلى صفيفتين جزئيتين بطول متساوٍ إذا كان ذلك ممكنًا. أي أن نجد نقطة المنتصف في الصفيفة الجزئية، ولنسمّها mid، وأن ندرس الصفيفتين الجزئيتين $A[low..mid]$ و $A[mid+1..high]$. يبيّن الشكل 4.4(أ) أن أية صفيفة جزئية متتالية $A[i..j]$ من $A[low..high]$ يجب أن تقع تمامًا في أحد المواقع التالية:

- محتواة بكاملها في الصفيفة الجزئية $A[low..mid]$، بحيث يكون $low \le i \le j \le mid$.
- محتواة بكاملها في الصفيفة الجزئية $A[mid+1..high]$، بحيث يكون $mid < i \le j \le high$، أو
- تحتوي نقطة المنتصف، بحيث يكون: $low \le i \le mid < j \le high$.

إذن، لا يمكن لصفيفة جزئية عظمى أن تقع إلا في أحد هذه المواقع. وفي الواقع، يجب أن يكون مجموع صفيفة جزئية عظمى من $A[low..high]$ هو الأكبر بين مجاميع كل الصفيفات الجزئية المحتواة كليًّا في $A[low..mid]$، والمحتواة كليًّا في $A[mid+1..high]$، أو التي تحتوي نقطة المنتصف. بإمكاننا أن نجد الصفيفتين الجزئيتين العظميَيْن لـ $A[low..mid]$ و $A[mid+1..high]$ عَوْديًّا، لأن هاتين المسألتين الجزئيتين هما منتسخان أصغر حجمًا من مسألة إيجاد الصفيفة الجزئية العظمى. وبهذا، يكون كل ما يتبقى علينا فعله هو إيجاد الصفيفة الجزئية ذات المجموع الأكبر من بين الصفيفات الثلاث التي تحتوي نقطة المنتصف.



الشكل 4.4 (أ) المواقع الممكنة للصفيفات الجزئية من $A[low..high]$: محتواة بأكملها في $A[low..mid]$، محتواة بأكملها في $A[mid+1..high]$، أو تحتوي نقطة المنتصف mid. **(ب)** أية صفيفة جزئية من $A[low..high]$ تحتوي نقطة المنتصف تكون مكونة من صفيفتين جزئيتين $A[i..mid]$ و $A[mid+1..j]$ حيث $low \le i \le mid$ و $mid < j \le high$.

يمكننا بسهولة إيجاد صفيفة جزئية عظمى تحتوي نقطة المنتصف في زمنٍ خطيٍّ متناسب مع حجم الصفيفة $A[low..high]$. إن هذه المسألة ليست منتسخًا أصغر من مسألتنا الأصلية، لأن فيها قيدًا إضافيًّا يتمثل في ضرورة أن تضم الصفيفة الجزئية المختارة نقطة المنتصف. وكما يبين الشكل 4.4(ب) فإن أية صفيفة جزئية تحتوي نقطة المنتصف هي في الحقيقة مركبة من صفيفتين جزئيتين $A[i..mid]$ و $A[mid+1..j]$، حيث $low \le i \le mid$ و $mid < j \le high$. لذا، فما علينا سوى إيجاد صفيفتين جزئيتين من الشكل $A[i..mid]$ و $A[mid+1..j]$ ثم ضمّهما معًا. إن دخل الإجراء FIND-MAX-CROSSING-SUBARRAY هو الصفيفة A والمؤشرات low و mid و $high$، ويعيد ثلاثيةً تحوي المؤشرين اللذين يحدّان صفيفة جزئية عظمى تضم نقطة المنتصف، إضافة إلى مجموع القيم في صفيفة جزئية عظمى.

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = -∞
 2  sum = 0
 3  for i = mid downto low
 4      sum = sum + A[i]
 5      if sum > left-sum
 6          left-sum = sum
 7          max-left = i
 8  right-sum = -∞
 9  sum = 0
10  for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

يعمل هذا الإجراء كما يلي. تجد السطور 1-7 صفيفة جزئية عظمى في النصف الأيسر، $A[low\,..\,mid]$. ولما كان واجبًا أن تضم هذه الصفيفة $A[mid]$، فإن حلقة **for** في السطور 3-7 يبدأ المؤشر i عند mid، ويتناقص حتى القيمة low، وبهذا تكون كل الصفيفات التي يدرسها هي من الشكل $A[i\,..\,mid]$. يعطي السطران 1-2 القيم البدائية للمتحولين: $left\text{-}sum$ الذي يحوي أكبر مجموع تحقق حتى الآن، و sum الذي يحوي العناصر في $A[i\,..\,mid]$. وكلما وجدنا، في السطر 5، صفيفة جزئية $A[i\,..\,mid]$ مجموعها أكبر من $left\text{-}sum$ ، نعدّل $left\text{-}sum$ إلى مجموع هذه الصفيفة الجزئية في السطر 6، ونعدّل في السطر 7 المتحوّل $max\text{-}left$ ليسجّل قيمة المؤشر i. تعمل السطور 8-14 بطريقة مماثلة على النصف الأيمن $A[mid+1\,..\,high]$. وهنا، يبدأ المؤشر j في حلقة **for** في السطور 10-14 من القيمة $mid+1$ ويتزايد حتى $high$، بحيث تكون كل صفيفة يدرسها من الشكل $A[mid+1\,..\,j]$. وأخيرًا يعيد السطر 15 المؤشرين $max\text{-}left$ و $max\text{-}right$ اللذين يحددان الصفيفة الجزئية التي تحتوي نقطة المنتصف، إضافة إلى المجموع $left\text{-}sum + right\text{-}$sum وهو مجموع القيم في الصفيفة الجزئية $A[max\text{-}left\,..\,max\text{-}right]$.

إذا كانت الصفيفة الجزئية $A[low\,..\,high]$ تحوي n عنصرًا (بحيث $n = high - low + 1$)، فإننا ندّعي أن الاستدعاء FIND-MAX-CROSSING-SUBARRAY$(A, low, mid, high)$ يستغرق زمنًا $\Theta(n)$. ولما كان كل تكرار لكلٍّ من حلقتي **for** يستغرق زمنًا $\Theta(1)$، فيكفي أن نَعُدّ عدد التكرارات الكلي. تقوم حلقة **for** في السطور 3-7 بإجراء $mid - low + 1$ تكرارًا، وحلقة **for** في السطور 10-14 بـ $high - mid$ تكرارًا، وبهذا يكون عدد التكرارات الكلي هو

$$\begin{aligned}(mid - low + 1) + (high - mid) &= high - low + 1 \\ &= n\,.\end{aligned}$$

الآن، وقد حصلنا على الإجراء ذي الزمن الخطي FIND-MAX-CROSSING-SUBARRAY، يمكننا أن نكتب شبه الرماز لخوارزميّة فرّق-تسد تحلّ مسألة الصفيفة الجزئية العظمى:

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
1  if high == low
2      return (low, high, A[low])          // base case: only one element
3  else mid = ⌊(low + high)/2⌋
4      (left-low, left-high, left-sum) =
           FIND-MAXIMUM-SUBARRAY(A, low, mid)
5      (right-low, right-high, right-sum) =
           FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
6      (cross-low, cross-high, cross-sum) =
           FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
```

```
8          return (left-low, left-high, left-sum)
9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10         return (right-low, right-high, right-sum)
11     else return (cross-low, cross-high, cross-sum)
```

سيجد الاستدعاء البدائي لـ FIND-MAXIMUM-SUBARRAY$(A, 1, A.length)$ صفيفة جزئية عظمى لـ $A[1..n]$.

يعيد الإجراء العَوْدي FIND-MAXIMUM-SUBARRAY، شأنه شأن الإجراء FIND-MAX-CROSSING-SUBARRAY، ثلاثيةً تضم مؤشرين يحدّان صفيفة جزئية عظمى، إضافة إلى مجموع القيم في هذه الصفيفة الجزئية العظمى. يختبر السطر 1 تحقّق الحالة الأساسيّة، حيث تكون الصفيفة الجزئية مكوّنة في عنصرٍ واحدٍ فقط. وعندما تكون الصفيفة الجزئية مكوّنة من عنصر واحد، فإن فيها صفيفة جزئية واحدة، هي نفسها، وبذلك يعيد السطر 2 ثلاثية فيها مؤشري البداية والنهاية لعنصر واحد فقط، إضافة إلى قيمته. تعالج السطور 3-11 الحالة العَوْدية. يقوم السطر 3 بجزء التقسيم، فيحسب مؤشر نقطة المنصف، وهو mid. نسمّي الصفيفة الجزئية $A[low..mid]$ ***الصفيفة الجزئية اليسرى left subarray*** والصفيفة $A[mid+1..high]$ ***الصفيفة الجزئية اليمنى right subarray***. ولما كنا نعرف أن الصفيفة الجزئية $A[low..high]$ تضم على الأقل عنصرين فإن كل من الصفيفتين الجزئيتَيْن اليسرى واليمنى ستضمان على الأقل عنصرًا واحدًا. يسود السطران 4 و 5 على المسألة من خلال الاستدعاء العَوْدي لإيجاد الصفيفتين الجزئيتين العظميَيْن في كلٍّ من الصفيفة الجزئية اليسرى ثمّ اليمنى. تقوم السطور 6-11 بجزء التجميع. يكتشف السطر 6 صفيفة جزئية عظمى بحيث تحوي نقطة المنتصف. (تذكَّرْ أننا نعتبر أن السطر 6 جزءًا من مرحلة تجميع الحل، لأنه يحلّ مسألة جزئية لا تمثّل منتسخًا أصغر من المسألة الأصلية.) فإذا دلَّ اختبار السطر 7 على أن الصفيفة الجزئية اليسرى تحتوي صفيفة جزئية ذات أكبر مجموع، فإن السطر 8 يعيد هذه الصفيفة الجزئية العظمى. وإلا فإن دلَّ اختبار السطر 9 على أن الصفيفة الجزئية اليمنى تحتوي صفيفة جزئية ذات أكبر مجموع، فإن السطر 10 يعيد هذه الصفيفة الجزئية العظمى. وإذا لم تحتوِ الصفيفة الجزئية اليسرى ولا اليمنى صفيفة جزئية تحقق أكبر مجموع، فمن المؤكَّد أن الصفيفة الجزئية العظمى تضم نقطة المنتصف، ويعيدها السطر 11.

تحليل خوارزمية فرّق–تسد

سنضع فيما يلي علاقة عَوْدية تصف زمن تنفيذ الإجراء العَوْدي FIND-MAXIMUM-SUBARRAY. سنعتمد، كما فعلنا عندما حللنا الفرز بالدمج في المقطع 2.3.2، الفرضية المبسّطة وهي أن حجم المسألة الأصلية هو من قوى 2، بحيث تكون أحجام كل المسائل الجزئية أعدادًا صحيحة. نرمز لزمن تنفيذ الإجراء FIND-MAXIMUM-SUBARRAY على صفيفة جزئية مؤلفة من n عنصرًا بـ $T(n)$. ونفترض، للمبتدئين، أن السطر 1 يستغرق زمنًا ثابتًا. وتكون الحالة القاعدية، عندما $n = 1$، سهلة: يستغرق السطر 2 زمنًا ثابتًا، إذن

$$T(1) = \Theta(1) . \quad (5.4)$$

تحدث الحالة العَوْدية عندما $n > 1$. يستغرق السطران 1 و 3 زمنًا ثابتًا. كل من المسألتين الجزئيتين المحلولتين في السطرين 6 و 5 هي مسألة على صفيفة جزئية من $n/2$ عنصرًا (تضمن لنا فرضيتنا بأن حجم المسألة الأصلية من قوى 2 أن $n/2$ هو عدد صحيح) ، وبهذا يستغرق حل كل منهما زمنًا $T(n/2)$. ولما كان علينا حل مسألتين جزئتين- للصفيفتين الجزئيتين اليسرى واليمنى — فإن مساهمة السطرين 4 و 5 في زمن التنفيذ تصل إلى $2T(n/2)$. وكما رأينا قبل قليل. فإن استدعاء FIND-MAX-CROSSING-SUBARRAY في السطر 6 يستغرق زمنًا $\Theta(n)$. تستغرق السطور 7-11 زمنًا $\Theta(1)$ فقط. وبهذا يكون لدينا في الحالة العَوْدية:

$$\begin{aligned} T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) \\ &= 2T(n/2) + \Theta(n) . \end{aligned} \quad (6.4)$$

وبدمج المعادلتين (5.4) و (6.4) نحصل على علاقة عَوْدية لـ $T(n)$ زمن تنفيذ الإجراء FIND-MAXIMUM-SUBARRAY:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases} \quad (7.4)$$

إن هذه العلاقة العَوْدية تماثل العلاقة العَوْدية في (1.4) للفرز بالدمج. وسنرى من الطريقة الرئيسة في المقطع 5.4 أن حل هذه العلاقة العَوْدية هو $T(n) = \Theta(n \lg n)$. ويمكنك أيضًا أن تعاود النظر إلى شجرة العَوْدية في الشكل 5.2 لتدرك لماذا ينبغي أن يكون الحل هو $T(n) = \Theta(n \lg n)$.

وهكذا نرى أن طريقة فرّق-تسد تعطي بالمقارنة خوارزمية أسرع من الطريقة الفجّة. فبعد أن عَرَفنا سابقًا طريقةَ الفرز بالدمج والآن عَرَفنا مسألةَ الصفيفة الجزئية العظمى، بدأنا نكوِّن فكرةً عن مدى قوّة طريقة فرّق-تسدْ. ستعطينا هذه الطريقة في بعض الأحيان أسرعَ الحلول لمسألةٍ ما بالمقارنة، وفي أحيان أخرى قد يكون بمقدورنا تحقيق حلٍّ أفضل. وكما يبيّن التمرين 1.4-5، هناك في الواقع خوارزمية ذات زمن خطي لمسألة الصفيفة الجزئية العظمى، وهي لا تستخدم مبدأ فرّق-تسد.

تمارين

1-1.4

ما الذي يعيده إجراء FIND-MAXIMUM-SUBARRY عندما تكون كل عناصر A سالبة؟

2-1.4

اكتب شبه رماز للطريقة الفجّة في حل مسألة الصفيفة الجزئية العظمى. يجب أن تُنفَّذ إجرائيتك في زمن $\Theta(n^2)$.

3-1.4

نَجِّز كلاً من الخوارزمية ذات الطريقة الفجّة والخوارزمية العَوْديّة لمسألة الصفيفة الجزئية العظمى على حاسوبك.

ما هو حجم المسألة n_0 الذي يمثل نقطة التجاوز التي تتفوق بدءًا منها الخوارزمية العَوْدية على خوارزمية الطريقة الفجّة؟ ثم غيّر الحالة القاعدية للخوارزمية العَوْدية لتستخدم الخوارزمية ذات الطريقة الفجّة كلّما كان حجم المسألة أقل من n_0. هل يغيّر هذا التعديل من نقطة التجاوز هذه؟

4-1.4

افترض أننا نغيّر تعريف مسألة الصفيفة الجزئية العظمى لتسمح للنتيجة بأن تكون صفيفة جزئية فارغة، حيث إن مجموع قيم صفيفة جزئية فارغة هو 0. كيف يمكنك أن تغيّر أيًّا من الخوارزميتين التي لا تسمح بصفيفات جزئية فارغة بأن تتيح أن تكون النتيجة صفيفة جزئية فارغة.

5-1.4

استخدم الأفكار التالية لتطوّر خوارزمية غير عَوْدية، ذات زمن خطي لحل مسألة الصفيفة الجزئية العظمى. ابدأ من النهاية اليسرى للصفيفة وتقدّم نحو اليمين، محتفظًا بالصفيفة الجزئية العظمى التي وقعت عليها قبل وصولك إلى المرحلة التالية. بمعرفتك الصفيفة الجزئية العظمى في $A[1..j]$، عمّمْ الجواب للعثور على الصفيفة الجزئية العظمى التي تنتهي بالمؤشر $j+1$ باستخدام الملاحظة التالية: إن صفيفة جزئية عظمى من $A[1..j+1]$ هي إما صفيفة جزئية عظمى من $A[1..j]$ وإما صفيفة جزئية من $A[i..j+1]$، لقيم مؤشر ما $1 \le i \le j+1$. حدّد صفيفة جزئية من الشكل $A[i..j+1]$ في زمن ثابت اعتمادًا على معرفتك لصفيفة جزئية عظمى تنتهي عند المؤشر j.

2.4 خوارزمية شتراسن لجداء المصفوفات

إذا كنت قد رأيت مصفوفاتٍ قبلاً، فربما أنت تعرف كيف تقوم بعملية جدائهما. (وإلاّ، فعليك أن تقرأ المقطع ت.1 في الملحق ث.) إذا كانت $A = (a_{ij})$ و $B = (b_{ij})$ مصفوفتين مربعتين $n \times n$، فإن جداءهما هو المصفوفة: $C = A \cdot B$، ونعرّف العنصر c_{ij} حيث $i,j = 1,2, \dots ,n$ بالعلاقة

$$c_{ij} = \sum_{k=1}^{n} a_{ij} \cdot b_{ij} \ , \tag{8.4}$$

يجب أن نحسب n^2 عنصرًا في المصفوفة، كلٌّ منها هو مجموع n قيمة. يأخذ الإجراء التالي مصفوفتين $n \times n$ هما A و B ويحسب جداءهما، معيدًا مصفوفة الجداء C ذات البعدين $n \times n$. نفترض أن لكل مصفوفةٍ واصفةً $rows$ تعطي عدد السطور فيها.

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
```

```
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik . b_kj
8  return C
```

يعمل الإجراء SQUARE-MATRIX-MULTIPLY كالتالي: تَحسب حلقةُ **for** في السطور 3-7 عناصرَ كل سطر i. وفي كل سطر i، تَحسب الحلقة **for** في السطور 4-7 كلَّ عنصر من العناصر c_{ij} لكل عمود j. يستبدئ السطر 5 قيمة c_{ij} بالصفر قبل أن نبدأ بحساب المجموع المعطى في المعادلة (8.4)، ويضيف كلُّ تكرار من الحلقة **for** في السطرين 6-7 حدًّا جديدًا إلى المعادلة (8.4).

لما كانت كلٌّ من حلقات **for** الثلاث المتداخلة تتكرر n تكرارًا تمامًا، ولما كان كل تنفيذ للسطر 7 يستغرق زمنًا ثابتًا، فإن الإجراء SQUARE-MATRIX-MULTIPLY يستغرق زمنًا $\Theta(n^3)$.

قد تعتقد بادئ الأمر أن أية خوارزمية لجداء المصفوفات يجب أن تستغرق زمنًا $\Omega(n^3)$، وذلك لأن التعريف الطبيعي لجداء المصفوفات يتطلب هذا العدد من عمليات الجداء. إلا أن اعتقادك هذا غير صحيح: إذ إن لدينا طريقة لحساب جداء المصفوفات في زمن $o(n^3)$. وسنرى في هذا المقطع خوارزمية شتراسن Strassen العَوْدية المتميِّزة لجداء مصفوفتين $n \times n$. تُنَفَّذ هذه الخوارزمية في زمن $\Theta(n^{\lg 7})$ سنبينّه في المقطع 5.4. ولما كانت قيمة lg 7 تقع بين 2.80 و 2.81، فإن خوارزمية شتراسن تُنَفَّذ في زمن $O(n^{2.81})$، وهذا أفضل مقارنةً بالإجراء البسيط SQUARE-MATRIX-MULTIPLY.

خوارزمية فرّق-تسد بسيطة

سنفترض، حتى نبقي الأمور بسيطة عندما نستخدم خوارزمية فرّق-تسد divide-and-conquer لحساب مصفوفة الجداء $C = A \cdot B$، أن n هي من قوى 2 الصحيحة في كل المصفوفات $n \times n$. نحن نتبنى هذه الفرضية لأننا في كل خطة تقسيم: ستقسم المصفوفات $n \times n$ إلى أربع مصفوفات $n/2 \times n/2$، وبافتراض أن n من قوى 2 الصحيحة، نضمن أن البعد $n/2$ هو عدد صحيح، مادام $n \geq 2$.

افترض أننا نقسم كلاً من المصفوفات A و B و C إلى أربع مصفوفات $n/2 \times n/2$.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}, \tag{9.4}$$

وبهذا نعيد كتابة المعادلة $C = A \cdot B$ بالشكل

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}. \tag{10.4}$$

توافق المعادلة (10.4) المعادلات الأربع التالية:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \ , \quad (11.4)$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \ , \quad (12.4)$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \ , \quad (13.4)$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \ . \quad (14.4)$$

تحدّد كلٌّ من هذه المعادلات الأربع جداءين لمصفوفات $n/2 \times n/2$، ومجموع ناتجي الجداء وهو مصفوفة $n/2 \times n/2$. يمكننا استخدام هذه المعادلات لإيجاد خوارزمية فرّق-تسد مباشرة عَوْدية:

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  if n == 1
4       c11 = a11 · b11
5  else partition A, B, and C in equations (9.4)
6       C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
7       C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
8       C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
9       C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
10 return C
```

إن شبه الرماز السابق يخفي تفصيل تنجيز دقيق وحيد، لكنه تفصيل هام. كيف نقسّم المصفوفات في السطر 5؟ فلو كان علينا إنشاء 12 مصفوفة جديدة $n/2 \times n/2$ لكنا أمضينا زمنًا $\Theta(n^2)$ في نسْخ العناصر. في الواقع، يمكننا تقسيم المصفوفات دون نسخ العناصر، والسر يكمن في الاعتماد على حساب المؤشرات. نحدّد مصفوفة جزئية بتحديد مجال من مؤشرات السطور ومجالٍ من مؤشرات الأعمدة في المصفوفة الأصلية. أي إننا نحصل في النهاية على تمثيل للمصفوفة الجزئية مختلف قليلاً عن طريقة تمثيل المصفوفة الأصلية، وهذا هو التفصيل الدقيق الذي نفعله في شبه الرماز. وفائدة ذلك هي أن تنفيذ السطر 5 لايستغرق إلا زمنًا $\Theta(1)$، إذ إننا نحدّد المصفوفات الجزئية من خلال حساب المؤشرات (ولكننا سنرى أنه سواء أقمنا بالنسخ أم بالتقسيم في المكان، فإن ذلك لا يؤثر بالمقارنة على زمن التنفيذ الكلي).

نشتق الآن علاقة عَوْدية توصّف زمن تنفيذ SQUARE-MATRIX-MULTIPLY-RECURSIVE. ليكن $T(n)$ الزمن اللازم لحساب جداء مصفوفتين $n \times n$ باستخدام هذا الإجراء. في الحالة القاعدية، عندما $n = 1$، نقوم فقط بعملية جداء سُلَّمي في السطر 4 وبهذا يكون

$$T(1) = \Theta(1) \ . \quad (15.4)$$

تحدث الحالة العَوْدية عندما يكون $n > 1$. وكما ناقشنا سابقًا، فإن تقسيم المصفوفات في السطر 5 يستغرق زمنًا $\Theta(1)$ باستخدام حساب المؤشرات. نقوم في السطور 6-9 باستدعاء SQUARE-MATRIX-MULTIPLY-RECURSIVE عَوْديًّا 8 مرات. ولما كان كلُّ استدعاءٍ يُحسب جداء مصفوفتين $n/2 \times n/2$، فإن مساهمته في زمن التنفيذ الكلي $T(n/2)$، ويكون الزمن الكلي الذي تستغرقه الاستدعاءات العَوْدية الثمانية هو $8T(n/2)$. ويجب أن نأخذ أيضًا في الحسبان عمليات جمع المصفوفات الأربع في السطور 6-9؛ فكلٌّ من هذه المصفوفات فيها $n^2/4$ عنصرًا، وبذلك فإن كلاًّ من عمليات جمع المصفوفات الأربع تستغرق زمنًا $\Theta(n^2)$. ولما كان عدد مرات جمع المصفوفات ثابتًا، فإن الزمن الكلي الذي يستغرقه جمع المصفوفات في السطور 6-9 هو $\Theta(n^2)$. (نستخدم هنا أيضًا حساب المؤشرات لنضع نتائج جمع المصفوفات في مواقعها الصحيحة داخل المصفوفة C، بزمن إضافي $\Theta(1)$ لكل عنصر.) إذن الزمن في الحالة العَوْدية هو مجموع زمن التقسيم وزمن جميع الاستدعاءات العَوْدية، وزمن جمع المصفوفات الناتجة عن الاستدعاءات العَوْدية:

$$\begin{aligned} T(n) &= \Theta(1) + 8T(n/2) + \Theta(n^2) \\ &= 8T(n/2) + \Theta(n^2)\ . \end{aligned} \tag{16.4}$$

لاحظ أنه إذا نَجّزنا التقسيم بنسخ المصفوفات، لاستغرق ذلك زمنًا $\Theta(n^2)$، ولما تغيّرت العلاقة العَوْدية، ولازداد زمن التنفيذ الكلي بمعامل ثابت فقط.

إن تجميع المعادلتين (15.4) و (16.4) يعطينا العلاقة لزمن تنفيذ SQUARE-MATRIX-MULTIPLY-RECURSIVE:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\ , \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1\ . \end{cases} \tag{17.4}$$

سنرى باستخدام الطريقة الرئيسة في المقطع 5.4 أن حل العلاقة العَوْدية (17.4) هو $T(n) = \Theta(n^3)$. أي إن طريقة فرّق-تسد البسيطة هذه ليست أسرع من إجراء SQUARE-MATRIX-MULTIPLY المباشر.

قبل أن نتابع دراسة خوارزمية شتراسن، سننظر من أين جاءت مركبات المعادلة (16.4). إن تقسيم كل مصفوفة $n \times n$ بحساب المؤشرات يستغرق زمنًا $\Theta(1)$، إلا أنه لدينا مصفوفتين يجب تقسيمهما. وعلى الرغم من أنه بإمكانك القول أن تقسيم مصفوفتين يستغرق $\Theta(2)$، إلا أن الثابت 2 متضمن في التدوين-Θ. إن جمع مصفوفتين، في كلٍّ منهما k عنصرًا، يستغرق زمنًا $\Theta(k)$. ولما كانت المصفوفات التي نجمعها ذات $n^2/4$ عنصرًا، يمكننا أن نقول إن جمع كل زوج من المصفوفات يستغرق زمنًا $\Theta(n^2/4)$، وهنا أيضًا يتضمن التدوين-Θ وجود المعامل الثابت 1/4، فنقول إن جمع مصفوفتين $n/2 \times n/2$ يستغرق زمنًا $\Theta(n^2)$. لدينا أربع عمليات جمع مثل هذه المصفوفات، وهنا أيضًا عوضًا عن أن نقول إن ذلك يستغرق $\Theta(4n^2)$، فإننا نكتفي بالقول إنها تستغرق زمنًا $\Theta(n^2)$. (طبعًا، قد تلاحظ أنه كان بإمكاننا القول إن جمع المصفوفات الأربع يستغرق زمنًا $\Theta(4n^2/4)$، وإن $4n^2/4 = n^2$، إلا أن الفكرة هنا أن التدوين-Θ يتضمن المعاملات

الثابتة، مهما كانت قيمتها.) وهكذا ننتهي مع حدين $\Theta(n^2)$ و $\Theta(1)$ يمكن تجميعهما في حدٍّ واحد.

أما عندما نحسب الاستدعاءات العَوْدية الثمانية، فليس بإمكاننا أن نعتبر المعامل الثابت 8 متضمنًا. بمعنى آخر، يجب أن نقول بوضوح إنها تستغرق معًا $8T(n/2)$ بدلاً من أن نكتفي بالقول $T(n/2)$. يمكنك أن تستشف ذلك بمعاودة النظر إلى شجرة العَوْدية في الشكل 5.2، للعلاقة العَوْدية (1.2) (وهي تماثل العلاقة العَوْدية (7.4))، مع الحالة العَوْدية $T(n) = 2T(n/2) + \Theta(n)$. حدَّد المعامل 2 عددَ الأبناء لكل عقدة في الشجرة، وهذا حدَّد بدوره عدد الحدود التي تدخل في المجموع في كل مستوى من الشجرة. لو كان لنا أن نتجاهل المعامل 8 في المعادلة (16.4) أو المعامل 2 في العلاقة العَوْدية (1.4) لأصبحت شجرة العَوْدية خطيّة فقط عوضًا عن أن تكون "كثيفة"، ولانحصرت مساهمة كل مستوى في المجموع في حدٍّ واحد فقط.

تذكَّر أنه على الرغم من أن التدوين المقارب يتضمن جداء المعاملات الثابتة، إلا أن التدوين العَوْدي مثل $T(n/2)$ لا يتضمنها أبدًا.

طريقة شتراسن

الفكرة الأساسية في طريقة شتراسن Strassen's method هي تقليل كثافة شجرة العَوْدية قليلاً؛ فعوضًا عن إجراء ثمانية جداءات عَوْدية لمصفوفات $n/2 \times n/2$ تُجرى سبعة فقط، ومقابل إلغاء جداء مصفوفات واحد، يجري المزيد من جمع المصفوفات، ولكن فقط عدد ثابت منها. وكما ذكرنا سابقًا، فإن عددًا ثابتًا من جمع المصفوفات سيكون متضمَّنًا في تدوين-Θ عندما نبني العلاقة العَوْدية التي توصّف زمن التنفيذ.

إن طريقة شتراسن ليست بديهية بتاتًا (قد يكون هذا التصريح هو الأخطر في هذا الكتاب) وهي مكونة من 4 خطوات:

1. نقسّم مصفوفات الدخل الدخل A و B ومصفوفة الخرج C إلى مصفوفات جزئية $n/2 \times n/2$، كما في المعادلة (9.4). تستغرق هذه الخطوة زمنًا $\Theta(1)$ بحساب المؤشرات تمامًا كما في الإجراء SQUARE-MATRIX-MULTIPLY.
2. ننشئ 10 مصفوفات $S_1, S_2, \ldots, S_{10}$ كلٌّ منها $n/2 \times n/2$، وهي ناتج جمع أو طرح مصفوفتين من بين تلك المنشأة في الخطوة 1. يمكن إنشاء هذه المصفوفات العشر كلها في زمن $\Theta(n^2)$.
3. باستخدام المصفوفات الجزئية المنشأة في الخطوة 1، والمصفوفات العشر المنشأة في الخطوة 2، نحسب عَوْديًّا 7 جداءات مصفوفات $P_1, P_2, \ldots, P_7$، كل مصفوفة P_i هي $n/2 \times n/2$.
4. نحسب المصفوفات الجزئية المطلوبة $C_{11}, C_{12}, C_{21}, C_{22}$ التي تكوّن المصفوفة C بجمع أو بطرح تركيبات مختلفة من المصفوفات P_i، يمكن حساب هذه المصفوفات الأربع جميعًا في زمن $\Theta(n^2)$.

سنرى تفاصيل الخطوات 2-4 بعد قليل، ولكن لدينا الآن ما يكفي من المعلومات لنبني العلاقة العَوْدية

التي تحكم زمن تنفيذ شتراسن. وقد افترضنا أنه ما إن يصل حجم المصفوفة n إلى 1، حتى نقوم بجداء سلمي بسيط، تمامًا كما في السطر 4 من SQUARE-MATRIX MULTIPLY-RECURSIVE. عندما يكون $n > 1$، تستغرق الخطوات 1 و 2 و 4 زمنًا مجمله $\Theta(n^2)$. وتتطلب الخطوة 3 إجراء سبع جداءات لمصفوفات $n/2 \times n/2$. وبهذا، نحصل على العلاقة العَوْدية التالية لزمن خوارزمية شتراسن $T(n)$:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\ , \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1\ . \end{cases} \tag{18.4}$$

لقد قايضنا جداءَ مصفوفةٍ واحدًا مقابل عددٍ ثابتٍ من عمليات جمع المصفوفات. وسنرى لاحقًا عندما نتعمَّق في معرفة العلاقات العَوْدية وحلولها أن هذه المقايضة تؤدي إلى زمن تنفيذ مقارب أدنى. واعتمادًا على الطريقة الرئيسة في المقطع 5.4، نجد أن حل العلاقة العَوْدية (18.4) هو $T(n) = \Theta(n^{\lg 7})$.

وفيما يلي تفصيل ذلك. ننشئ في الخطوة 2 المصفوفات العشر التالية:

$$\begin{aligned}
S_1 &= B_{12} - B_{22}\ , \\
S_2 &= A_{11} + A_{12}\ , \\
S_3 &= A_{21} + A_{22}\ , \\
S_4 &= B_{21} - B_{11}\ , \\
S_5 &= A_{11} + A_{22}\ , \\
S_6 &= B_{11} + B_{22}\ , \\
S_7 &= A_{12} - A_{22}\ , \\
S_8 &= B_{21} + B_{22}\ , \\
S_9 &= A_{11} - A_{21}\ , \\
S_{10} &= B_{11} + B_{12}\ .
\end{aligned}$$

ولما كان علينا أن نجمع أو نطرح مصفوفات $n/2 \times n/2$ عشر مرات، فإن هذه الخطوة تستغرق في الحقيقة زمنًا $\Theta(n^2)$.

نُجري، في الخطوة 3، سبع جداءات عَوْديًّا لمصفوفات $n/2 \times n/2$ لحساب المصفوفات التالية، التي يَنتج كل منها عن مجموع أو فرق جداءات لمصفوفات جزئيّة من A و B:

$$\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22}\ , \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22}\ , \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11}\ , \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11}\ , \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}\ , \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}\ , \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}\ .
\end{aligned}$$

لاحظ أن عمليات الجداء الوحيدة التي نحتاج إلى إنجازها هي العمليات الموجودة في العمود الأوسط من المعادلات السابقة. أما العمود الأيمن فهو لمجرد أن يبيِّن ماذا تساوي هذه الجداءات بدلالة المصفوفات الجزئية الأصليّة المنشأة في الخطوة 1.

تقوم الخطوة 4 بجمع وطرح المصفوفات P_i المحسوبة في الخطوة 3 لتبني المصفوفات الجزئية $n/2 \times n/2$ الأربع التي تكوّن مصفوفة الجداء C. نبدأ بالمصفوفة

$$C_{11} = P_5 + P_4 - P_2 + P_6 \ .$$

وبنشر الطرف الأيمن، بتعويض كل P_i بسطرها الخاص وبوضع الحدود التي يَحذف أحدُها الآخر في العمود نفسه، نجد أن C_{11} تساوي:

$$\begin{array}{l}
A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
\qquad\qquad\qquad - A_{22} \cdot B_{11} \qquad\qquad + A_{22} \cdot B_{21} \\
\qquad - A_{11} \cdot B_{22} \qquad\qquad\qquad\qquad\qquad - A_{12} \cdot B_{22} \\
\qquad\qquad\qquad\qquad - A_{22} \cdot B_{22} - A_{22} \cdot B_{21} + A_{12} \cdot B_{22} - A_{12} \cdot B_{21} \\
\hline
A_{11} \cdot B_{11} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + A_{12} \cdot B_{21} \ ,
\end{array}$$

وهذا يتوافق مع المعادلة (11.4).

بالمثل نضع $C_{12} = P_1 + P_2$ ، فنجد أن C_{12} تساوي:

$$\begin{array}{l}
A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
\qquad + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
\hline
A_{11} \cdot B_{12} \qquad\qquad + A_{12} \cdot B_{22} \ ,
\end{array}$$

وهذا يتوافق مع المعادلة (12.4).

وبوضع $C_{21} = P_3 + P_4$، نجد أن C_{21} تساوي:

$$\begin{array}{l}
A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
\qquad - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\
\hline
A_{21} \cdot B_{11} \qquad\qquad + A_{22} \cdot B_{21} \ ,
\end{array}$$

وهذا يتوافق مع المعادلة (13.4).

وأخيرًا، نضع $C_{22} = P_5 + P_1 - P_3 - P_7$، فنجد أن C_{22} تساوي:

$$\begin{array}{l}
A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
\qquad - A_{11} \cdot B_{22} \qquad\qquad\qquad\qquad + A_{11} \cdot B_{12} \\
\qquad\qquad\qquad - A_{22} \cdot B_{11} \qquad\qquad\qquad\qquad - A_{21} \cdot B_{11} \\
-A_{11} \cdot B_{11} \qquad\qquad\qquad\qquad\qquad - A_{11} \cdot B_{12} + A_{21} \cdot B_{11} + A_{21} \cdot B_{12} \\
\hline
\qquad\qquad\qquad\qquad A_{22} \cdot B_{22} \qquad\qquad\qquad\qquad + A_{21} \cdot B_{12} \ ,
\end{array}$$

وهذا يتوافق مع المعادلة (14.4). وبالجملة، فإننا نقوم في الخطوة 4 بجمع وطرح مصفوفات $n/2 \times n/2$ ثماني مرات، وبذلك تستغرق هذه الخطوة فعليًا زمنًا $\Theta(n^2)$.

وهكذا نرى أن خوارزمية شتراسن، المكونة من الخطوات 1-4، تعطي جداء المصفوفات الصحيح، وأن العلاقة العَوْدية (18.4) توصّف زمن تنفيذها. ولما كان حل هذه العلاقة العَوْدية هو $T(n) = \Theta(n^{\lg 7})$، كما سنرى في المقطع 5.4، فإن طريقة شتراسن أسرع بالمقاربة من إجراء الجداء المباشر SQUARE-MATRIX-MULTIPLY. تناقش الملاحظات المذكورة في نهاية الفصل بعض الجوانب العملية لخوارزمية شتراسن.

تمارين

ملاحظة: على الرغم من أن التمارين 2.4-3، و 2.4-4، و 2.4-5 تتعلق بخوارزميات معدّلة عن خوارزمية شتراسن، فمن المستحسن قراءة المقطع 5.4 قبل أن تحاول حلّها.

1-2.4

استخدم خوارزمية شتراسن لحساب جداء المصفوفتين

$$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix} .$$

مبينًا تفاصيل عملك.

2-2.4

اكتب شبه رماز لخوارزمية شتراسن.

3-2.4

كيف يمكنك تغيير خوارزمية شتراسن لحساب جداء مصفوفتين $n \times n$ حيث لا تكون n من قوى 2 الصحيحة؟ بيّن أن الخوارزمية الناتجة تُنفَّذ في زمن $\Theta(n^{\lg 7})$.

4-2.4

ما هو أكبر عدد k بحيث، إذا كان بإمكاننا إجراء جداء مصفوفتين 3×3 باستخدام k جداءً (دون الاستفادة من تبادلية الجداء)، يكون بإمكاننا إجراء جداء المصفوفات $n \times n$ في زمن $o(n^{\lg 7})$؟ كيف يكون زمن تنفيذ هذه الخوارزمية؟

5-2.4

اكتشفَ V. Pan طريقةً لجداء مصفوفتين 68×68 باستخدام 132,464 جداءً، وطريقةً لجداء مصفوفتين 70×70 باستخدام 143,640 جداءً، وطريقةً لجداء مصفوفتين 72×72 باستخدام 155,424 جداءً. أي الطرق تعطي أفضل زمن تنفيذ مقارب عند استخدامها في خوارزمية فرِّق-تسد لجداء المصفوفات؟ كيف تتقارن هذه الخوارزمية بخوارزمية شتراسن؟

6-2.4

ما سرعة تنفيذ جداء مصفوفة $kn \times n$ بمصفوفة $n \times kn$ باستخدام خوارزمية ستراشن إجراءً فرعيًّا؟ أجب عن السؤال نفسه بقلب ترتيب مصفوفتَي الدخل.

7-2.4
بيّن كيف تُجري عملية جداء العددين العقديين $a + bi$ و $c + di$ باستخدام ثلاث عمليات جداء أعداد حقيقيّة فقط. ينبغي أن يكون دخل الخوارزمية الأعداد a، و b، و c، و d، وأن تعيد المركّبة الحقيقية $ac-bd$ والمركبة التخيليّة $ad + bc$ كلاً منهما على حدة.

3.4 طريقة التعويض لحل العلاقات العَوْدية

بعد أن رأينا كيف توصِّف العلاقاتُ العَوْدية أزمنةَ تنفيذ خوارزميات فرّق-تسد، سنتعلم كيف نحل هذه العلاقات العَوْدية. نبدأ في هذا المقطع بطريقة "التعويض".

تتضمن ***طريقة التعويض substitution method*** المستخدَمة في حل العلاقات العَوْدية خطوتين:

1. تخمين أسلوب الحل.
2. استخدام الاستقراء الرياضي للعثور على الثوابت، وبيان صحة الحل.

يَعود اسم الطريقة إلى أننا نعوِّض الحلَّ المخمَّن للدالّة عندما نطبّق فرضية الاستقراء على قيم أصغر. هذه الطريقة فعّالة، إلا أنه يجب أن نكون قادرين على تخمين شكل الجواب حتى نتمكن من استخدامها.

يمكن استخدام طريقة التعويض لإيجاد حدود عليا أو دنيا للعلاقة العَوْديّة. وكمثال على ذلك، نحدِّد حدًّا أعلى للعلاقة العَوْديّة

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \,, \qquad (19.4)$$

التي تشبه العلاقات العَوْدية (3.4) و (4.4). نخمن أن الحل هو $T(n) = O(n \lg n)$. تتطلب طريقة التعويض أن نبرهن أن $T(n) \le cn \lg n$ عند اختيار مناسب للثابت $c > 0$. نبدأ بافتراض أن هذا الحدّ محقَّق عند كل الأعداد الموجبة $m < n$، وخاصة عند $m = \lfloor n/2 \rfloor$، وهذا يعطي $T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$.
وبالتعويض داخل العلاقة العَوْدية نحصل على

$$\begin{aligned} T(n) &\le 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\ &\le cn \lg(n/2) + n \\ &\le cn \lg n - cn \lg 2 + n \\ &\le cn \lg n - cn + n \\ &\le cn \lg n \,, \end{aligned}$$

حيث تكون الخطوة الأخيرة محقَّقة مادام $c \ge 1$.

يفرض الاستقراء الرياضي علينا الآن أن نبيّن أن حلّنا محقَّق عند الشروط الحديّة. وعادة ما نقوم بذلك بأن نبيّن أن الشروط الحدّيّة مناسبة باعتبارها حالات أساسيّة بالنسبة للبرهان بالاستقراء. إذن، يجب أن نبيّن

في حالة العلاقة العَوْديّة (19.4) أنه بإمكاننا اختيار الثابت c كبيرًا كفاية بحيث يكون الحدّ $T(n) \leq cn \lg n$ محققًا أيضًا في حالة الشروط الحديّة. وقد يؤدي هذا المطلب في بعض الأحيان إلى بعض المشكلات. لنفترض، من أجل المناقشة، أن $T(1) = 1$ هو الشرط الحدّي الوحيد للعلاقة العَوْديّة. فإذا كان $n = 1$، فإن الحدّ $T(n) \leq cn \lg n$ يعطي المتراجحة $T(1) \leq c1 \lg 1 = 0$، وهذا يناقض $T(1) = 1$. وبناء على ذلك، فإن الحالة الأساسية لبرهاننا بالاستقراء غير محققة.

يمكننا تجاوز هذه الصعوبة في برهان فرضيّة الاستقراء على شرط حدّي محدَّد ببذل القليل من المجهود الإضافي. ففي العلاقة العَوْدية (19.4) مثلاً، نستفيد من أن التدوين المقارب لا يتطلَّب سوى أن نبرهن أن $T(n) < cn \lg n$ عندما $n \geq n_0$، حيث n_0 ثابت نختاره. نحتفظ بالشرط الحدّي الذي يسبب لنا المشاكل $T(1) = 1$، إلا أننا لا نأخذه في الحسبان في البرهان بالاستقراء. نقوم بذلك بأن نلاحظ أنه عندما $n > 3$، لا تعتمد العلاقة العَوْدية مباشرة على $T(1)$. وهكذا، يمكننا استبدال $T(2)$ و $T(3)$ بـ $T(1)$ باعتبارهما حالات أساسية في برهاننا بالاستقراء، ونضع $n_0 = 2$. لاحظ أننا نميّز بين الحالة الأساسيّة للعلاقة العَوْديّة ($n = 1$)، والحالتين الأساسيّتين للبرهان بالاستقراء ($n = 2$ و $n = 3$). باستخدام $T(1) = 1$، نستنتج من العلاقة العَوْديّة أن $T(2) = 4$ و $T(3) = 5$. ويمكن الآن استكمال البرهان بالاستقراء للعلاقة $T(n) \leq cn \lg n$ لثابتٍ ما $c \geq 1$، بأن نختار c كبيرة كفاية بحيث يكون $T(2) \leq c2 \lg 2$ و $T(3) \leq c3 \lg 3$. وكما يبدو لنا، فإن أيَّ اختيارٍ لـ $c \geq 2$ كافٍ للحالتين الأساسيّتين $n = 2$ و $n = 3$. سيكون من السهل توسيع الشروط الحديّة لمعظم العلاقات العَوْديّة التي سندرجها لجعل فرضية الاستقراء سارية عندما تكون قيم n صغيرة، ولن نوضّح هذا النوع من التفاصيل دائماً.

صياغة تخمين جيد

لا يوجد، لسوء الحظ، طريقة عامة لتخمين الحلول الصحيحة للمعادلات العَوْديّة. يتطلب تخمين الحل خبرةً، وأحيانًا بعض الإبداع. ولكن، لحسن الحظ، هناك بعض الطرق الكسبيّة heuristics التي يمكن أن تساعدك لتصبح مخمنًا جيدًا. يمكنك أيضًا استخدام شجرات العَوْدية، التي سنراها في المقطع 4.4 لتوليد تخمينات جيدة.

إذا كانت العلاقة العَوْديّة مشابهة لعلاقةٍ صادفتك من قبل، فإن تخمين حلٍّ مشابه سيكون معقولاً. مثال: لنأخذ العلاقة العَوْدية

$$T(n) = 2T(\lfloor n/2 \rfloor + 17) + n \ ,$$

التي تبدو صعبة بسبب القيمة "17" المضافة في محدد T في الطرف الأيمن. إلا أننا بالحدس نرى أن هذا الحد الإضافي غير قادر على التأثير تأثيرًا جوهريًّا على حل العلاقة العَوْديّة. عندما تكون n كبيرة، لن يكون الفارق بين $\lfloor n/2 \rfloor$ و $\lfloor n/2 \rfloor + 17$ كبيرًا: كلاهما يقسم n إلى قسمين متعادلين تقريبًا. ومن ثَمَّ، نخمن أن الحل هو

$T(n) = O(n \lg n)$، وهذا ما يمكنك التحقق منه باستخدام طريقة التعويض (انظر التمرين 3.4-6).

هناك طريقة أخرى للحصول على تخمين جيّد تتمثل في برهان حدود عليا ودنيا غير ملاصقة للعلاقة العَوْديّة، ثم السعي لتقليل مجال الريبة. مثلاً، قد نبدأ بحدٍّ أدنى للعلاقة العَوْدية (19.4) من الشكل $T(n) = \Omega(n)$، وذلك لأن العلاقة العَوْدية فيها الحد n، ويمكننا البرهان على حد أعلى مبدئي هو $T(n) = O(n^2)$. بعد ذلك، نقوم تدريجيًا بتخفيض الحدّ الأعلى ورفع الحدّ الأدنى حتى نتقارب إلى الحلّ الصحيح الملاصق بالمقاربة، وهو $T(n) = \Theta(n \lg n)$.

حالات تتطلب دقةً

في بعض الأحيان، يكون بإمكانك تخمين حدٍّ مقارب صحيح لحلّ العلاقة العَوْديّة، إلاّ أنه يبدو أن العمليات الرياضيّة لا تسير على ما يرام في برهان الاستقراء. وعادة ما تكمن المشكلة في أنّ فرضيّة الاستقراء ليست بالقوّة الكافية لبرهان الحدّ بتفاصيله. وعندما تواجه مثل هذه العقبة، كثيرًا ما تسمح مراجعة التخمين عن طريق طرح حدٍّ من مرتبة أدنى من الحلّ المخمّن بأن تسير العمليات الرياضيّة كما يجب.

لندرسْ العلاقة العَوْدية

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \ .$$

نخمّن أن الحل هو $T(n) = O(n)$، ونحاول أن نبيّن أن $T(n) \le cn$ مع اختيار مناسب لـ c. بتعويض تخميننا في العلاقة العَوْديّة، نحصل على

$$\begin{aligned} T(n) &\le c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \end{aligned}$$

وهذا لا يقتضي أن $T(n) \le cn$ مهما كان اختيار c. وقد نرغب في محاولة حلٍّ مخمّن أكبر، وليكن $T(n) = O(n^2)$. ومع أنه يمكننا التحقُّق من هذا التخمين الأكبر، فإن تخميننا الأصلي أن الحل هو $T(n) = O(n)$ صحيح. وحتى نبيّن ذلك، علينا في الواقع أن نضع فرضيّة استقراء أقوى.

نرى بالحدس أن تخميننا صحيح تقريبًا: فنحن بعيدون عنه بمقدار الثابت 1، وهو حد من درجة أصغر. إلاّ أن الاستقراء الرياضي لا يتحقق حتى نبرهن الصيغة الدقيقة لفرضية الاستقراء. سنتجاوز هذه الصعوبة بأن *نطرح* من تخميننا السابق حدًّا من مرتبة أصغر، فيصبح تخميننا الجديد $T(n) \le cn - d$ حيث $d \ge 0$ ثابت. لدينا الآن

$$\begin{aligned} T(n) &\le (c\lfloor n/2 \rfloor - d) + (c\lceil n/2 \rceil - d) + 1 \\ &= cn - 2d + 1 \\ &\le cn - d \ , \end{aligned}$$

مادام $d \ge 1$. ومثلما سبق، يجب اختيار الثابت c كبيرًا كفاية ليحقق الشروط الحديّة.

قد تجد فكرة طرح حدّ من مرتبة أصغر غير بديهية. ولكن في النهاية، إذا كانت العمليات الرياضيّة لا تتم كما يجب، ألا يجدر بنا أن نزيد تخميننا قليلاً؟ ليس بالضرورة! عندما نستخدم الاستقراء الرياضي لبرهان حد أعلى، قد يكون في الواقع برهان حدٍّ أضعف أكثر صعوبة، لأننا نحتاج عند برهان هذا الحد الأضعف إلى استخدام الحد الأضعف نفسه عَوْديًّا في البرهان: في مثالنا الحالي، وعندما تتضمن العلاقة العَوْدية أكثر من حد عَوْدي واحد، قمنا بطرح الحد ذي المرتبة الدنيا من الحد المقترح مرةً لكل حد عَوْدي. ففي المثال السابق، طرحنا الثابت d مرتين: مرةً للحد $T(\lfloor n/2 \rfloor)$، ومرة للحد $T(\lceil n/2 \rceil)$. فانتهى بنا الأمر إلى المتراجحة $T(n) \leq cn - 2d + 1$، وكان من السهل إيجاد قيم لـ d تجعل $cn - 2d + 1$ أصغر من $cn - d$ أو تساويه.

تحاشي العثرات

من السهل الوقوع في الخطأ عند استخدام التدوين المقارب. بإمكاننا مثلاً، في العلاقة العَوْديّة (19.4)، أن نبرهن خطأً أن $T(n) = O(n)$ بأن نخمّن $T(n) \leq cn$ وأن نناقش

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor) + n \\ &\leq cn + 1 \\ &= O(n), \qquad \Leftarrow \text{!! خطأ} \end{aligned}$$

إذ إن c ثابت. والخطأ هو أننا لم نبرهن *الصيغة الدقيقة* لفرضية الاستقراء، وهي $T(n) \leq cn$. إذن، سنبرهن صراحةً أن $T(n) \leq c$ عندما نريد أن نبين أن $T(n) = O(n)$.

تغيير المتحولات

في بعض الأحيان، قد تساعد عملية جبريّة صغيرة على أن تجعل علاقةً عَوْديّة غير معروفة مشابهةً لأخرى رأيتَها قبلاً. مثلاً، لتكن لدينا العلاقة العَوْديّة

$$T(n) = 2T\left(\lfloor \sqrt{n} \rfloor\right) + \lg n ,$$

التي تبدو صعبة. إلا أنه يمكننا أن نبسط هذه العلاقة العَوْديّة باستخدام تغيير المتحولات. وللتبسيط، لن نهتم بأمر تدوير القيم مثل $\sqrt{n}$ لتكون طبيعيّة. وبإعادة تسمية $m = \lg n$ نحصل على

$$T(2^m) = 2T\left(2^{m/2}\right) + m .$$

يمكننا الآن إعادة تسمية $S(m) = T(2^m)$ لنحصل على العلاقة العَوْدية الجديدة

$$S(m) = 2S(m/2) + m ,$$

التي تشابه كثيرًا العلاقة (19.4). وفي الحقيقة فإن العلاقة التكرارية الجديدة لها الحل نفسه

$S(m) = O(m \lg m)$. وبالعَوْدة من $S(m)$ إلى $T(n)$، نحصل على

$$T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \ \lg \lg n) \ .$$

تمارين

1-3.4

بيّن أن حل العلاقة $T(n) = T(n-1) + n$ هو $O(n^2)$.

2-3.4

بيّن أن حل العلاقة $T(n) = T(\lceil n/2 \rceil) + 1$ هو $O(\lg n)$.

3-3.4

رأينا أن حلّ $T(n) = 2T(\lfloor n/2 \rfloor) + n$ هو $O(n \lg n)$. بيّن أن حل هذه العلاقة العَوْدية هو أيضًا $\Omega(n \lg n)$. استنتج أن الحل هو $\Theta(n \lg n)$.

4-3.4

بيّنْ أنه يمكننا بتغييّر فرضية الاستقراء، أن نتجاوز المشكلة المتعلقة بالشرط الحدي $T(1) = 1$ في العلاقة العَوْدية (19.4) دون أن نضبط الشروط الحديّة المتعلقة بالبرهان بالاستقراء.

5-3.4

بيّنْ أن $\Theta(n \lg n)$ هو الحل للعلاقة الحديّة "الدقيقة" (3.4) للفرز بالدمج.

6-3.4

بيّنْ أن حل العلاقة $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ هو $O(n \lg n)$.

7-3.4

يمكنك باستخدام الطريقة الرئيسة في المقطع 5.4 أن تبيّن أن حلّ العلاقة العَوْديّة $T(n) = 4T(n/3) + n$ هو $T(n) = \Theta(n^{\log_3 4})$. بيّن أن طريقة التعويض غير قادرة على برهان الفرضية $T(n) \le cn^{\log_3 4}$. ثم بيّن كيف يسمح طرح حد أدنى مرتبة بإتمام البرهان بالتعويض.

8-3.4

يمكنك باستخدام الطريقة الرئيسة في المقطع 5.4 أن تبيّن أن حلّ العلاقة العَوْديّة $T(n) = 4T(n/2) + n$ هو $T(n) = \Theta(n^2)$. بيّن أن طريقة التعويض غير قادرة على برهان الفرضية $T(n) \le cn^2$. ثم بيّن كيف يسمح طرح حد أدنى مرتبة بإتمام البرهان بالتعويض.

9-3.4

حُلّ العلاقة العَوْديّة $T(n) = 3T(\sqrt{n}) + \log n$ بإجراء تغيير في المتحولات. يجب أن يكون حلّك ملاصقًا بالمقاربة. ولا تعبأ بكون القيم طبيعية أو لا.

4.4 طريقة شجرة العَوْديّة لحل العلاقات العَوْدية

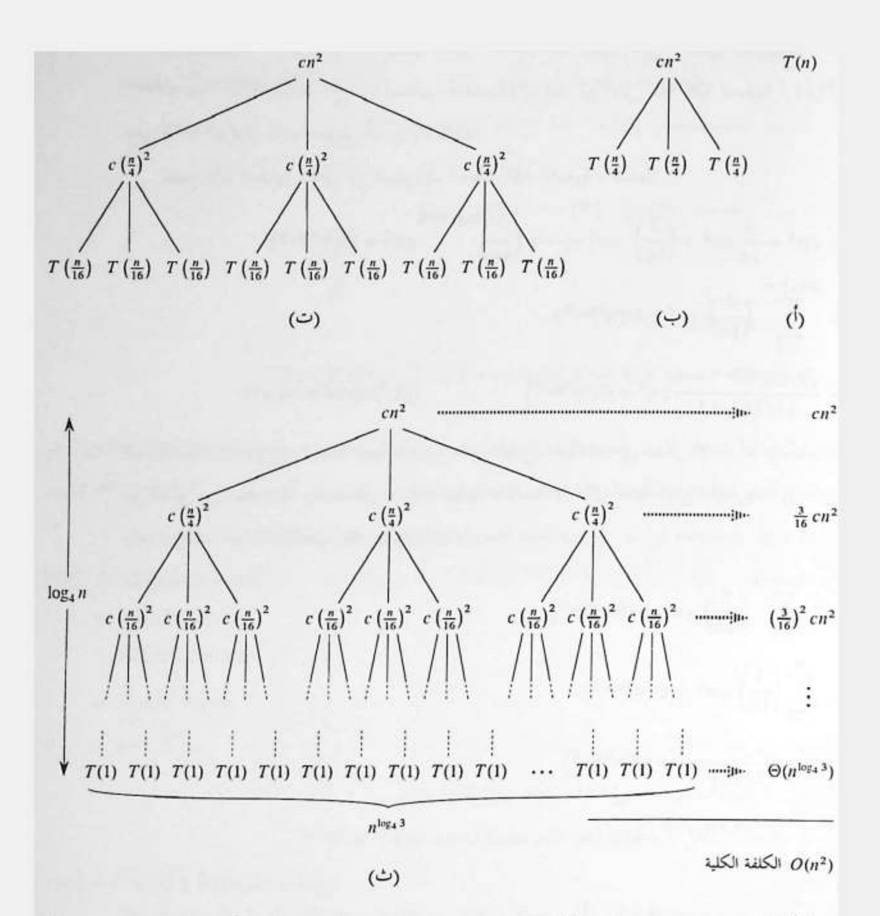
على الرغم من أن بمقدورك استخدام طريقة التعويض لنقدّم برهانًا موجزًا لصحة حل علاقة عَوْدية ما، إلا أنه قد يصعب عليك في بعض الأحيان التكهن بتخمين جيّد، وعندها يعتبر رسم شجرة عَوْديّة، كما فعلنا في تحليلنا للعلاقة العَوْدية للفرز بالدمج في المقطع 2.3.2، طريقةً مباشرة للخروج بتخمين جيّد. تُمثّل كل عقدة في ***شجرة العَوْدية recursion tree***، كلفةَ مسألة جزئية واحدة موجودة في مكان ما داخل مجموعة الاستدعاءات العَوْديّة. نجمع الكلف داخل كل مستوى من الشجرة لنحصل على مجموعة من التكاليف وفق المستويات، ثم نجمع هذه الكلف كلّها لنحسب الكلفة الكلّية لكل المستويات في الاستدعاءات العَوْدية.

أحسن ما تقوم به شجرة العَوْديّة هي أنها تولّد تخمينًا جيدًا، ليجري التحقق منه فيما بعد باستخدام طريقة التعويض. عندما تَستخدم شجرة العَوْدية للحصول على تخمين جيّد، يمكنك دائمًا التسامح مع قدر قليل من "عدم الدقة"، إذ إنك ستقوم لاحقًا بالتحقق من تخمينك. ولكن إذا كنت حريصًا جدًّا وأنت ترسم شجرة العَوْديّة، وتجمع التكاليف، يمكنك استخدامها برهانًا مباشرًا لحل العلاقة العَوْدية. سنستخدم شجرات العَوْدية، في هذا المقطع لتوليد تخمينات جيدة، وسنستخدمها في المقطع 6.4 لنبرهن مباشرة المبرهنة التي تكوّن الأساس للطريقة الرئيسة.

سنجد، على سبيل المثال، كيف يمكن لشجرة عَوْدية أن تعطي تخمينًا جيدًا للعلاقة العَوْدية $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$. نبدأ بالتركيز على البحث عن حدٍّ أعلى للحل. ولما كنا نعرف أنه لا أثر عادة لدوال الأسقف والأرضيات في حل العلاقات العَوْدية (هذا مثال على عدم الدقة الذي يمكننا القبول به هنا)، فإننا سننشئ شجرة العَوْدية للعلاقة العَوْدية $T(n) = 3T(n/4) + cn^2$ معلنين بأن المعامل الثابت المستخدم هو $c > 0$.

يبيّن الشكل 5.4 اشتقاق شجرة العَوْدية للعلاقة $T(n) = 3T(n/4) + cn^2$. وللسهولة، نفترض أن n من قوى 4 (مثال آخر عن عدم الدقة المقبول) بحيث تكون كل أحجام المسائل الجزئية أعدادًا صحيحة. يبيّن الجزء (أ) من الشكل: $T(n)$، التي تُوسَّع في الجزء (ب) إلى شجرة مكافئة تمثّل العلاقة العَوْدية. يمثّل الحد cn^2 في الجذر الكلفة في أعلى مستوى من الاستدعاء العَوْدي، وتمثل الأشجار الفرعية الثلاث المتفرعة من الجذر التكاليف الناتجة عن المسائل الجزئية التي قياسها $n/4$. يبيّن الجزء (ت) هذه العمليّة مكرّرة خطوة أخرى بنشر كل عقدة كلفتها $T(n/4)$ من الجزء (ب). وكلفة كل من الأبناء الثلاثة للجذر هي $c(n/4)^2$. ونتابع نشر كل عقدة في الشجرة بأن نفرّعها إلى الأجزاء التي تتكون منها تبعًا للعلاقة العَوْديّة.

لمّا كانت أحجام المسائل الجزئية تتناقص بمعامل 4 كلما نزلنا من مستوى إلى المستوى الأدنى منه، كان لا بدّ لنا من الوصول إلى شرطٍ حدّي. ولكن كم يبعد الجذر عن مثل هذا الحدّ؟ إن حجم المسألة الجزئية لعقدةٍ عمقها i هو $n/4^i$، إذن سيصل حجم المسألة الجزئية إلى $n = 1$ عندما يصبح $n/4^i = 1$، أو ما يكافئه من أن $i = \log_4 n$. إذن يكون للشجرة $\log_4 n + 1$ مستوى (على الأعماق $0, 1, 2, \ldots, \log_4 n$).



الشكل 5.4 بناء شجرة عَوْديّة للعلاقة $T(n) = 3T(n/4) + cn^2$. يبيّن الجزء (أ) $T(n)$ التي ننشرها تدريجيًا في الأجزاء (**ب**)–(**ث**) لنكوّن شجرة العَوْديّة. يبلغ ارتفاع الشجرة المنشورة كليًّا في الجزء (**ث**) $\log_4 n$ (وفيها $\log_4 n + 1$ مستوى).

بعد ذلك نحدّد كلفة كل مستوى من الشجرة. يضم كل مستوى ثلاثة أضعاف العُقد الموجودة في المستوى الذي يعلوه، وبذلك يكون عدد العقد على العمق i مساويًا 3^i. ولما كان حجم المسائل الجزئية ينقص وفق عامل يساوي 4 لكل مستوى نَنْزل إليه بدءًا من الجذر، تكون كلفة كل عقدة على العمق i، عندما $i = 0, 1, 2, \ldots, \log_4 n - 1$، مساوية $c(n/4^i)^2$. وبضرب هذه الكلفة بعدد العقد في المستوى i، تكون الكلفة الكلية على العمق i، عندما $i = 0, 1, 2, \ldots, \log_4 n - 1$، مساوية

لـ $3^i c(n/4^i)^2 = (3/16)^i cn^2$ ويضم أدنى المستويات، وهو على العمق $\log_4 n$، ما مقداره $3^{\log_4 n} = n^{\log_4 3}$ عقدة، كل منها يساهم بكلفة $T(1)$، إذن من أجل كلفة كلّيّة مساوية لـ $n^{\log_4 3}T(1)$، وهي $\Theta(n^{\log_4 3})$، إذ إننا نفترض أن $T(1)$ ثابت.

نجمع الآن التكاليف على كل المستويات لنحدّد كلفة الشجرة بأكملها:

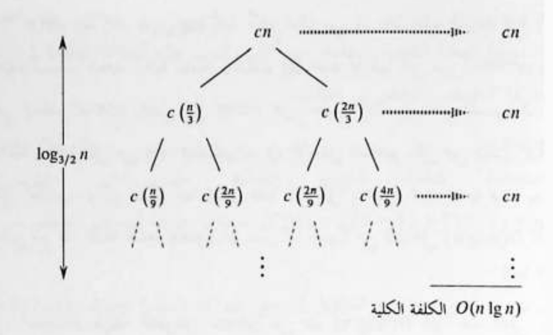
$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n-1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}) \qquad \text{((باستخدام المعادلة (أ.5))}
\end{aligned}
$$

تبدو الصيغة الأخيرة التي توصلنا إليها فوضوية بعض الشيء، وذلك حتى نتذكر مجددًا أنه بإمكاننا الاستفادة من القليل من عدم الدقة ونستخدم سلسلة هندسية متناقصة لا نهائية كحدٍّ أعلى. فإذا عدنا إلى الوراء خطوة واحدة وطبقنا المعادلة (الملحق أ.6)، يكون لدينا:

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2) \ .
\end{aligned}
$$

إذن، اشتققنا تخمينًا بأن $T(n) = O(n^2)$ من علاقتنا العَوْدية الأصلية $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$. تشكّل معاملات الحدّ cn^2 في هذا المثال سلسلة هندسية متناقصة، وباستخدام المعادلة في (الملحق أ.6)، يمكن حدّ مجموع هذه المعاملات من الأعلى بالثابت 16/13. ولما كانت مساهمة الجذر في الكلفة الكلية هي cn^2، فإن الجذر يساهم بجزء ثابت من هذه الكلفة. وبعبارة أخرى، تسيطر كلفة الجذر على الكلفة الكلية للشجرة.

إنّ، $O(n^2)$ هو في الواقع حدُّ أعلى للعلاقة العَوْدية، وهذا ما سنتحقق منه بعد قليل، إذن يجب أن يكون هذا الحد ملاصقًا. لماذا؟ يساهم الاستدعاء العَوْدي الأول بكلفة $\Theta(n^2)$، إذًا يجب أن يكون $\Omega(n^2)$ حدًّا أدنى للعلاقة العَوْدية.



الشكل 6.4 شجرة عَوْديّة للعلاقة العَوْدية $T(n) = T(n/3) + T(2n/3) + cn$.

بمقدورنا الآن أن نستخدم طريقة التعويض للتحقق من أن تخميننا صحيح، أي إن $T(n) = O(n^2)$ هو حد أعلى للعلاقة العَوْدية $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$. نريد أن نبيّن أن $T(n) \le dn^2$ لثابت $d > 0$. باستخدام الثابت $c > 0$ نفسه الذي استخدمناه من قبل، يكون لدينا

$$\begin{aligned} T(n) &\le 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\le 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\le 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\le dn^2 , \end{aligned}$$

حيث تكون الخطوة الأخيرة محققة مادام $d \ge (16/3)c$.

يبيّن الشكل 6.4 مثالاً آخر أشد تعقيدًا لشجرة العَوْدية للعلاقة

$$T(n) = T(n/3) + T(2n/3) + O(n) .$$

(نحذف، ثانيةً، دالّتَي السقف والأرضية بهدف التبسيط.) وكما سبق، نجعل c يمثل العامل الثابت في الحدّ $O(n)$. وعندما نجمع القيم لمستويات شجرة العَوْدية المبينة في الشكل، نحصل على القيمة cn لكل مستوى من المستويات. وأطول مسار بسيط من الجذر إلى ورقة هو $n \to (2/3)n \to (2/3)^2 n \to \cdots \to 1$. ولما كان $(2/3)^k n = 1$ عندما $k = \log_{3/2} n$، فإن ارتفاع الشجرة يساوي $\log_{3/2} n$.

نتوقع بالحدس أن يكون حل العلاقة العَوْدية هو على الأكثر عدد المستويات مضروبًا بكلفة كل مستوى، أو $O(cn \log_{3/2} n) = O(n \log n)$. يبين الشكل 6.4 المستويات العليا فقط من شجرة العَوْدية، ولا تساهم كل المستويات بكلفة cn. لنأخذ كلفة الأوراق. لو كانت شجرة العَوْدية هذه شجرة ثنائية كاملة ارتفاعها $\log_{3/2} n$، لكان هناك $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$ ورقة. ولما كانت كلفة كل ورقة ثابتةً، وجب أن تكون الكلفة

الكلية للأوراق $\Theta(n^{\log_{3/2} 2})$، ولما كان $\log_{3/2} n$ ثابتًا أكبر تمامًا من 1، فإن هذه الكلفة هي $\omega(n \lg n)$. ولكن شجرة العَوْدية هذه ليست شجرة ثنائية كاملة، ولذلك فإن عدد أوراقها أقل من $n^{\log_{3/2} 2}$ ورقة. أضف إلى ذلك أن مزيدًا من العقد الداخلية يختفي مع ابتعادنا عن الجذر. إذن، تساهم المستويات الأقرب إلى الأسفل في الكلفة الكليّة بكلفة أقل من cn. فالمستويات في الأسفل تساهم بأقل من ذلك. يمكننا البحث عن حساب دقيق لكل الكلف، ولكن تذكّرْ أننا نحاول فقط أن نأتي بتخمين نستخدمه فيما بعد في طريقة التعويض، فدعْنا نتسامح مع قلة الدقة هذه، ونحاول أن نبيّن أن تخمينًا من الشكل $O(n \lg n)$ للحدِّ الأعلى هو تخمين صحيح.

يمكننا في الواقع، استخدام طريقة التعويض لنتحقق من أن $O(n \lg n)$ هو حد أعلى لحلّ العلاقة العَوْدية. نبيّن فيما يلي أن $T(n) \le d\, n \lg n$، حيث d ثابت موجب مناسب. لدينا

$$
\begin{aligned}
T(n) &\le T(n/3) + T(2n/3) + cn \\
&\le d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= (d(n/3)\lg n - d(n/3)\lg 3) \\
&\qquad + (d(2n/3)\lg n - d(2n/3)\lg(3/2)) + cn \\
&= d\, n \lg n - d\big((n/3)\lg 3 + (2n/3)\lg(3/2)\big) + cn \\
&= d\, n \lg n - d\big((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2\big) + cn \\
&= d\, n \lg n - d\, n(\lg 3 - 2/3) + cn \\
&\le d\, n \lg n \,,
\end{aligned}
$$

مادام $d \ge c/(\lg 3 - (2/3))$، إذن، لم نكن مضطرين لإجراء حساب أكثر دقة للكلف في شجرة العَوْدية.

تمارين

1-4.4

استخدمْ شجرة العَوْدية لتحديد حد أعلى مقارب جيّد للعلاقة العَوْدية $T(n) = 3T(\lfloor n/2 \rfloor) + n$. استخدم طريقة التعويض للتحقق من جوابك.

2-4.4

استخدمْ شجرة العَوْدية لتحديد حد أعلى مقارب جيّد للعلاقة العَوْدية $T(n) = T(n/2) + n^2$. استخدم طريقة التعويض للتحقق من جوابك.

3-4.4

استخدمْ شجرة العَوْدية لتحديد حد أعلى مقارب جيّد للعلاقة العَوْدية $T(n) = 4T(n/2 + 2) + n$. استخدم طريقة التعويض للتحقق من جوابك.

4-4.4

استخدمْ شجرة العَوْدية لتحديد حد أعلى مقارب جيّد للعلاقة العَوْدية $T(n) = 2T(n-1) + 1$. استخدم طريقة التعويض للتحقق من جوابك.

5-4.4

استخدمْ شجرة العَوْدية لتحديد حد أعلى مقارب جيّد للعلاقة العَوْدية $T(n) = T(n-1) + T(n/2) + n$. استخدم طريقة التعويض للتحقق من جوابك.

6-4.4

برهن باستخدام شجرة العَوْدية أن حل العلاقة العَوْدية $T(n) = T(n/3) + T(2n/3) + cn$، حيث c ثابت، هو $\Omega(n \lg n)$.

7-4.4

ارسم شجرة العَوْدية للعلاقة $T(n) = 4T(\lfloor n/2 \rfloor) + cn$، حيث c ثابت وأعط حدًّا مقاربًا ملاصقًا لحلّها. تحقّقْ – باستخدام طريقة التعويض – من الحدّ الذي تقترحه.

8-4.4

استخدم شجرة العَوْدية لإعطاء حل ملاصق بالمقاربة للعلاقة العَوْدية $T(n) = T(n-a) + T(a) + cn$، حيث $a \geq 1$ و $c > 0$ ثوابت.

9-4.4

استخدم شجرة العَوْدية لإعطاء حل ملاصق بالمقاربة للعلاقة العَوْدية $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$ ، حيث α ثابت محصور في المجال $0 < \alpha < 1$ و $c > 0$ ثابت أيضًا.

5.4 الطريقة الرئيسة لحل العلاقات العَوْدية

تقدّم الطريقة الرئيسة master method "وصفةً" لحل العلاقات العَوْديّة من الشكل

$$T(n) = aT(n/b) + f(n) , \qquad (20.4)$$

حيث $a \geq 1$ و $b > 1$ ثابتان و $f(n)$ دالة موجبة بالمقاربة. لاستخدام الطريقة الرئيسة، تحتاج إلى تذكّر ثلاث حالات، يمكنك بعدها حل العديد من العلاقات العَوْديّة بسهولة مطلقة، غالبًا بدون ورقة وقلم.

تصف العلاقة العَوْدية في (20.4) زمن تنفيذ خوارزمية تقسم مسألةً حجمها n إلى a مسألة جزئية، كل منها حجمه n/b، حيث a و b ثابتان موجبان. تُحلُّ المسائل الجزئية التي عددها a عَوْديًّا، كل منها في زمن $T(n/b)$. وتصف الدالة $f(n)$ كلفة تقسيم المسألة وكلفة تجميع نتائج المسائل الجزئية. مثلاً تأخذ العلاقة

العَوْدية الناتجة عن خوارزمية شتراسن القيم $a = 7$ و $b = 2$ و $f(n) = \Theta(n^2)$.

إن العلاقة العَوْدية غير معرّفة جيدًا من وجهة نظر الصحّة التقنية، إذ قد لا يكون n/b عددًا طبيعيًّا. إلا أن الاستعاضة عن كل من الحدود $T(n/b)$ والتي عددها a، سواءً بـ $T(\lfloor n/b \rfloor)$ أو بـ $T(\lceil n/b \rceil)$ لا يؤثر على السلوك المقارب للعلاقة العَوْديّة. (سنبرهن هذه الفرضية في المقطع القادم.) ولذلك عادةً ما نجد حذف دالتي الأرضية والسقف مناسبًا عند كتابة العلاقات العَوْديّة "فرّقْ-تسدْ" من هذا الشكل.

المبرهنة الرئيسة

تعتمد الطريقة الرئيسة على المبرهنة التالية:

مبرهنة 1.4 (المبرهنة الرئيسة)

ليكن $a \geq 1$ و $b > 1$ ثابتين، ولتكن $f(n)$ دالة، ولتكن $T(n)$ معرّفة على الأعداد الطبيعيّة بالعلاقة العَوْدية

$$T(n) = a\,T(n/b) + f(n) \ ,$$

حيث نفسّر n/b على أنها $\lfloor n/b \rfloor$) أو $\lceil n/b \rceil$. وعندها يكون لـ $T(n)$ الحدود التالية بالمقاربة:

1. إذا كان $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ حيث $\epsilon > 0$ ثابت ما، فإن $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. إذا كان $f(n) = \Theta\left(n^{\log_b a}\right)$، فإن $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$.
3. إذا كان $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ حيث $\epsilon > 0$ ثابت ما، وإذا كان $af(n/b) \leq cf(n)$ حيث $c < 1$ ثابت ما، و n كبيرة كفاية، فإن $T(n) = \Theta(f(n))$. ■

دعونا، قبل أن نطبق المبرهنة الرئيسة على بعض الأمثلة، نمضي بعض الوقت في محاولة فهم هذه المبرهنة. نحن نقارن في كلٍّ من الحالات الثلاث، الدالةَ $f(n)$ بالدالةِ $n^{\log_b a}$. يتحدّد حل العلاقة العَوْدية حدسيًّا تبعًا لأكبر الدالتين. إذا كانت الدالة $n^{\log_b a}$ هي الأكبر، كما في الحالة 1، فإن الحل هو $T(n) = \Theta\left(n^{\log_b a}\right)$. وإذا كانت الدالة $f(n)$ هي الأكبر، كما في الحالة 3، فإن الحل هو $T(n) = \Theta(f(n))$. وإذا كانت الدالتان من القياس نفسه، كما في الحالة 2، فإننا نضرب بعامل لوغاريتمي، ويكون الحل $T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta(f(n) \lg n)$.

يقف بيننا وبين هذا الحدس، أن نكون واعين لبعض التفاصيل التقنية. ففي الحالة الأولى، لا يكفي أن تكون $f(n)$ أصغر من $n^{\log_b a}$، بل يجب أن تكون أصغر منها *حدوديًّا polynomially*. أي أن تكون $f(n)$ أصغر بالمقاربة من $n^{\log_b a}$ بعامل n^{ϵ} حيث $\epsilon > 0$ ثابت. ولا يكفي في الحالة الثالثة أن تكون الدالة $f(n)$ أكبر من $n^{\log_b a}$، بل يجب أن تكون أكبر منها حدوديًّا، ويجب أن تحقق شرط "الانتظام" أي $af(n/b) \leq cf(n)$. وعمومًا تحقّق هذا الشرط معظم الدوال المحدودة حدوديًّا التي سندرسها.

لاحظ أن الحالات الثلاث لا تشمل كل حالات $f(n)$ الممكنة؛ فهناك فجوة تفصل بين الحالتين 1 و 2، عندما تكون $f(n)$ أصغر من $n^{\log_b a}$، ولكن ليس أصغر حدوديًا. وبالمثل هناك فجوة تفصل بين الحالتين 2 و 3، عندما تكون $f(n)$ أكبر من $n^{\log_b a}$ ولكن ليست أكبر حدوديًا. إذا وقعت الدالة $f(n)$ في إحدى هاتين الفجوتين الفاصلتين، أو إذا لم يتحقق شرط الانتظام في الحالة 3، لا يمكنك استخدام الطريقة الرئيسة لحل العلاقة العَوْدية.

استخدام الطريقة الرئيسة

لاستخدام الطريقة الرئيسة، نحدّد من المبرهنة الرئيسة أية حالة (إن وُجدت) تنطبق ونكتب الجواب.

لنأخذ كمثال أول

$$T(n) = 9T(n/3) + n \ .$$

لدينا، في حالة هذه العلاقة العَوْدية، $a = 9$ و $b = 3$ و $f(n) = n$، وبهذا لدينا $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$. ولما كان $f(n) = O(n^{\log_3 9 - \epsilon})$، حيث $\epsilon = 1$، يمكننا تطبيق الحالة 1 من المبرهنة الرئيسة، ونستنتج أن الحل هو $T(n) = \Theta(n^2)$.

لنأخذ الآن

$$T(n) = T(2n/3) + 1 \ ,$$

حيث $a = 1$ و $b = 3/2$ و $f(n) = 1$ و $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$. تنطبق الحالة 2، لأن $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$، وهكذا يكون حل العلاقة العَوْدية $T(n) = \Theta(\lg n)$.

لدينا في حالة العلاقة العَوْدية

$$T(n) = 3T(n/4) + n \lg n \ ,$$

$a = 3$ و $b = 4$ و $f(n) = n \lg n$ و $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$، ولما كان $f(n) = \Omega(n^{\log_4 3 + \epsilon})$، حيث $\epsilon \approx 0.2$، فإن الحالة 3 تنطبق إذا استطعنا أن نبيّن أن $f(n)$ تحقق شرط الانتظام. فإذا كانت n كبيرة كفاية، يكون لدينا $af(n/b) = 3(n/4)\lg(n/4) \le (3/4)n \lg n = cf(n)$، حيث $c = 3/4$. وهكذا يكون حل العلاقة العَوْديّة اعتمادًا على الحالة 3: $T(n) = \Theta(n \lg n)$.

إن الطريقة الرئيسة لا تنطبق على العلاقة العَوْدية

$$T(n) = 2T(n/2) + n \lg n \ ,$$

رغم أن لها الصيغة المناسبة: $a = 2$ و $b = 2$ و $f(n) = n \lg n$ و $n^{\log_b a} = n$. قد تعتقد خطأً أن الحالة 3 يجب أن تنطبق، لأن $f(n) = n \lg n$ أكبر من $n^{\log_b a} = n$ بالمقاربة، ولكن المشكلة أنها ليست أكبر *حدوديًا*. فالنسبة $f(n)/n^{\log_b a} = (n \lg n)/n = \lg n$ و أصغر من n^{ϵ} بالمقاربة مهما كانت قيمة الثابت الموجب ϵ. ومن ثم تقع العلاقة العَوْدية في الفجوة الفاصلة بين الحالة 2 والحالة 3. (انظر التمرين 2-6.4

لإيجاد حل لها.)

لنستخدم الطريقة الرئيسية لحل العلاقات العَوْدية التي رأيناها في المقطعين 1.4 و 2.4. إن العلاقة العَوْدية (7.4)

$$T(n) = 2T(n/2) + \Theta(n) ,$$

توصِّف أزمنة تنفيذ فرّق-تسد لكلٍّ من مسألة الصفيفة الجزئية العظمى والفرز بالدمج. (كما هو الحال دائمًا، أهملنا ذكر الحالة القاعدية في العلاقة العَوْدية.) لدينا هنا $a = 2$ و $b = 2$ و $f(n) = \Theta(n)$، وبذلك يكون لدينا $n^{\log_b a} = n^{\log_2 2} = n$. تنطبق الحالة 2، لأن $f(n) = \Theta(n)$، وبذلك يكون لدينا الحل $T(n) = \Theta(n \lg n)$.

أما العلاقة العَوْدية (17.4)

$$T(n) = 8T(n/2) + \Theta(n^2) ,$$

فتوصِّف زمن تنفيذ خوارزمية فرّق-تسد الأولى التي رأيناها لجداء المصفوفات. لدينا الآن $a = 8$ و $b = 2$، و $f(n) = \Theta(n^2)$، وبذلك يكون لدينا $n^{\log_b a} = n^{\log_2 8} = n^3$. ولما كانت n^3 أكبر من $f(n)$ حدوديًّا (أي $f(n) = O(n^{3-\epsilon})$ حيث $\epsilon = 1$)، فإن الحالة 1 تنطبق، ويكون $T(n) = \Theta(n^3)$.

وأخيرًا، لنأخذ العلاقة العَوْدية (18.4)

$$T(n) = 7T(n/2) + \Theta(n^2) ,$$

التي تصف زمن تنفيذ خوارزمية شتراسن. لدينا هنا $a = 7$، و $b = 2$، و $f(n) = \Theta(n^2)$، وبذلك يكون لدينا $n^{\log_b a} = n^{\log_2 7}$. بإعادة كتابة $\log_2 7$ على أنها $\lg 7$، وبتذكر أن $2.80 < \lg 7 < 2.81$ نرى أن $f(n) = O(n^{\lg 7-\epsilon})$ حيث $\epsilon = 0.8$. وتنطبق الحالة 1 ثانيةً، ويكون لدينا الحل $T(n) = \Theta(n^{\lg 7})$.

تمارين

1-5.4

استخدم الطريقة الرئيسة لتعطي حدودًا ملاصقة بالمقاربة للعلاقات العَوْدية التالية:

أ. $T(n) = 2T(n/4) + 1$.

ب. $T(n) = 2T(n/4) + \sqrt{n}$.

ت. $T(n) = 2T(n/4) + n$.

ث. $T(n) = 2T(n/4) + n^2$.

2-5.4

يرغب الأستاذ قيصر Caesar بتطوير خوارزمية جداء مصفوفات أسرع من خوارزمية شتراسن بالمقارنة.

ستستخدم خوارزميته طريقة فرّق-تسد، مقسّمة إلى أجزاء قياسها $n/4 \times n/4$، وستستغرق خطوات التفريق والتجميع معًا زمنًا $\Theta(n^2)$. يحتاج لتحديد عدد المسائل الجزئية التي يجب أن تنشئها خوارزميته حتى يتفوق على خوارزمية شتراسن. إذا أنشأتْ خوارزميته a مسألة جزئية، فإن العلاقة العَوْدية لزمن التنفيذ $T(n)$ تصبح $T(n) = aT(n/4) + \Theta(n^2)$. ما هي أكبر قيمة صحيحة لـ a تكون بها خوارزمية الأستاذ قيصر أسرع من خوارزمية شتراسن بالمقاربة؟

3-5.4

استخدم الطريقة الرئيسة لتبيّن أن حل العلاقة العَوْديّة للبحث الثنائي $T(n) = T(n/2) + \Theta(1)$ هو $T(n) = \Theta(\lg n)$. (انظر التمرين 5-3.2 لوصف البحث الثنائي.)

4-5.4

هل يمكن تطبيق الطريقة الرئيسة على العلاقة العَوْدية $T(n) = 4T(n/2) + n^2 \lg n$؟ علّلْ. أعطِ حدًّا أعلى بالمقاربة لهذه العلاقة العَوْدية.

5-5.4 ★

لنأخذ شرط الانتظام $af(n/b) \le cf(n)$ حيث $c < 1$ ثابت ما، الذي هو جزء من الحالة 3 من المبرهنة الرئيسة. أعطِ مثالاً على ثابتين $a \ge 1$ و $b > 1$ ودالة $f(n)$ تحقق كل الشروط في الحالة 3 من المبرهنة الرئيسة ما عدا شرط الانتظام.

★ 6.4 برهان المبرهنة الرئيسة

يتضمن هذا المقطع برهان المبرهنة الرئيسة (المبرهنة 1.4). ولا حاجة إلى أن تفهم البرهان لكي تطبق هذه المبرهنة.

إن البرهان يتألف من جزأين. يحلّل الجزء الأوّل العلاقة العَوْدية "الرئيسة master" (20.4) مع الافتراض المبسّط أن $T(n)$ معرّفة فقط عند القوى الصحيحة لـ $b > 1$، أي عندما $n = 1, b, b^2, \ldots$. ويعطي هذا الجزء كل الحدس اللازم للاقتناع بصحة المبرهنة الرئيسة. ويبيّن الجزء الثاني كيف يمكن تعميم التحليل السابق على كل الأعداد الطبيعيّة n، وذلك بتطبيق تقنيات رياضيّة لمسألة التعامل مع الأرضيات والأسقف.

سنسرف في هذا المقطع، بعض الشيء في استخدام التدوين المقارب، وذلك باستخدامه أحيانًا لوصف سلوك دوال معرّفة على قوى b الصحيحة فقط. تذكّرْ أن تعاريف التدوينات المقاربة تتطلب برهان الحدود لكل الأعداد الكبيرة كفاية، وليس لقوى b فقط. ولكن، لما كان بمقدورنا وضع تدوينات مقاربة جديدة تنطبق على المجموعة $\{b^i : i = 0, 1, 2, \ldots\}$، عوضًا عن الأعداد الطبيعيّة، فإن هذا التجاوز ضئيل الأهميّة.

ومع ذلك، يجب أن نكون دائمًا متنبهين عندما نستخدم التدوين المقارب على نطاق محدود حتى لا

نتوصل إلى استنتاجات خاطئة. فعلى سبيل المثال، إذا برهنا أن $T(n) = O(n)$ عندما تكون n قوة صحيحة لـ 2، فإن هذا لا يضمن أن يكون $T(n) = O(n)$. إذ يمكن تعريف الدالة $T(n)$ كما يلي

$$T(n) = \begin{cases} n & \text{if } n = 1, 2, 4, 8, \dots, \\ n^2 & \text{otherwise} , \end{cases}$$

حيث إن أفضل حد أعلى يمكن برهانه هنا هو $T(n) = O(n^2)$. لذلك، وبسبب هذا النوع من العواقب السيئة، فإننا لن نستخدم التدوين المقارب أبدًا على نطاق محدود دون أن نشير إلى ذلك بوضوح تام في السياق.

1.6.4 البرهان في حالة القوى الصحيحة

يحلّل الجزء الأول من برهان المبرهنة الرئيسة العلاقة العَوْدية (20.4)

$$T(n) = aT(n/b) + f(n) ,$$

الواردة في الطريقة الرئيسة، مع الفرض بأن n هي قوّة صحيحة لـ $b > 1$، حيث ليس بالضرورة أن تكون b عددًا طبيعيًّا. ينقسم التحليل إلى ثلاث توطئات. تختزل الأولى مسألة حل العلاقة العَوْدية العامة إلى مسألة حساب عبارة فيها مجموع. وتحدّد التوطئة الثانية حدودًا على هذا المجموع. وتجمع التوطئة الثالثة سابقتيها معًا لتبرهن نسخة من المبرهنة الرئيسة عندما تكون n إحدى القوى الصحيحة لـ b.

توطئة 2.4

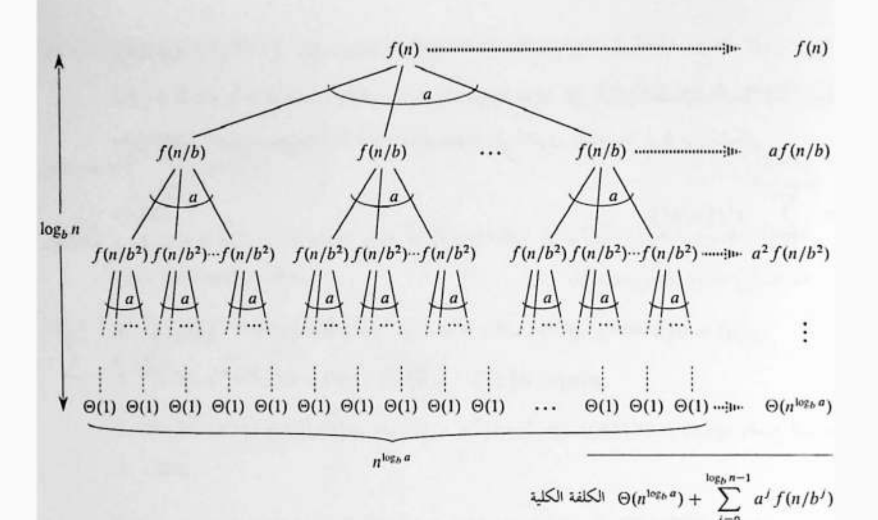
ليكن $a \geq 1$ و $b > 1$ ثابتين، ولتكن $f(n)$ دالة موجبة معرّفة على القوى الصحيحة لـ b. نعرّف $T(n)$ على القوى الصحيحة لـ b بالعلاقة العَوْدية

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ aT(n/b) + f(n) & \text{if } n = b^i , \end{cases}$$

حيث i عدد صحيح موجب. إذن

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) . \qquad (21.4)$$

البرهان نستخدم شجرة العَوْدية المذكورة في الشكل 7.4. إن كلفة جذر الشجرة هي $f(n)$، وله a ابنًا، كلفة كلٍّ منهم $f(n/b)$. (من المناسب أن نتعامل مع a على أنه عدد طبيعي، وخاصة عندما نعاين شجرة العَوْدية، إلا أن ذلك غير ضروري رياضيًّا.) ولكل من هؤلاء الأبناء a ابنًا، وهكذا هناك a^2 عقدة على البعد 2 من الجذر، كلفة كل منها $f(n/b^2)$. وفي الحالة العامة، هناك a^j عقدة على البعد j من الجذر، وكل منها كلفته $f(n/b^j)$. كلفة كل ورقة هي $T(1) = \Theta(1)$، وكل ورقة هي على عمق $\log_b n$، وذلك لأن $n/b^{\log_b n} = 1$. وهناك $a^{\log_b n} = n^{\log_b a}$ ورقة في الشجرة.



الشكل 7.4 شجرة العَوْديّة التي تولّدها العلاقة $T(n) = a\,T(n/b) + f(n)$. إن هذه الشجرة هي شجرة كاملة ذات a فرعًا من كل عقدة مع $n^{\log_b a}$ ورقة، ارتفاعها $\log_b n$. نجد إلى اليمين من كل مستوى كلفته، وتعطي المعادلة (21.4) مجموع هذه التكاليف.

يمكن الوصول إلى المعادلة (21.4) بجمع تكاليف كل المستويات في الشجرة، كما هو مبيّن في الشكل. كلفة العقد الداخلية على العمق j هو $a^j f(n/b^j)$ وهكذا يكون المجموع لكل المستويات الداخلية

$$\sum_{j=0}^{\log_b n-1} a^j f(n/b^j) .$$

يمثل هذا المجموع، في خوارزمية فرّق-تسد التي أنتجته، تكاليف تقسيم المسائل إلى مسائل جزئية ومن ثمّ تجميع هذه المسائل الجزئية. وتكون كلفة كل الأوراق $\Theta(n^{\log_b a})$، وهي كلفة حلّ $n^{\log_b a}$ مسألةً جزئيةً حجمُ كلٍّ منها 1. ■

تقابل الحالات الثلاث في المبرهنة الرئيسة، بالنظر إلى شجرة العَوْدية الحالات التي تكون فيها الكلفة الكلية للشجرة (1) جلّها من كلف الأوراق (2) موزّعة بالتساوي في جميع مستويات الشجرة أو (3) جلّها في كلفة الجذر.

يصف المجموع في العلاقة (21.4) كلفة خطوات التقسيم والتجميع في خوارزمية فرّق-تسد التي أنتجت العلاقة. وتزوّدنا المبرهنة التالية بحدود مقاربة لنمو هذا المجموع.

توطئة 3.4

ليكن $a \geq 1$ و $b > 1$ ثابتين، ولتكن $f(n)$ دالة موجبة معرّفة على القوى الصحيحة لـ b. يمكن إعطاء حدّ مقارب على القوى الصحيحة لـ b لدالة $g(n)$ معرّفة على القوى الصحيحة لـ b من الشكل

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) \tag{22.4}$$

وذلك تبعًا للحالات التالية:

1. إذا كان $f(n) = O(n^{\log_b a-\epsilon})$، حيث $\epsilon > 0$ ثابت ما، فإن $g(n) = O(n^{\log_b a})$.
2. إذا كان $f(n) = \Theta(n^{\log_b a})$، فإن $g(n) = \Theta(n^{\log_b a} \lg n)$.
3. إذا كان $af(n/b) \leq cf(n)$، حيث $c < 1$ ثابت ما، فإن $g(n) = \Theta(f(n))$، لجميع قيم n الكبيرة كفاية.

البرهان لدينا في الحالة 1: $f(n) = O(n^{\log_b a-\epsilon})$، وهذا يقتضي أن $f(n/b^j) = O((n/b^j)^{\log_b a-\epsilon})$. بالتعويض في المعادلة (22.4) نجد

$$g(n) = O\left(\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a-\epsilon}\right) . \tag{23.4}$$

نحد هذا المجموع في التدوين-O بتفريق الحدود وبالتبسيط، فينتج عن ذلك سلسلة هندسية متزايدة:

$$\begin{aligned}
\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a-\epsilon} &= n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} \left(\frac{ab^\epsilon}{b^{\log_b a}}\right)^j \\
&= n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} (b^\epsilon)^j \\
&= n^{\log_b a-\epsilon} \left(\frac{b^{\epsilon \log_b n}-1}{b^\epsilon-1}\right) \\
&= n^{\log_b a-\epsilon} \left(\frac{n^\epsilon-1}{b^\epsilon-1}\right) .
\end{aligned}$$

ولما كان b وϵ ثابتين، فباستطاعتنا إعادة كتابة العبارة الأخيرة كالتالي $n^{\log_b a-\epsilon} O(n^\epsilon) = O(n^{\log_b a})$. وبتعويض هذه العبارة بالمجموع في المعادلة (23.4) نجد

$$g(n) = O(n^{\log_b a}) ,$$

وبهذا نكون قد برهنا الحالة 1.

ولما كانت الحالة 2 تفترض أن $f(n) = \Theta(n^{\log_b a})$، فإن $f(n/b^j) = \Theta((n/b^j)^{\log_b a})$. وبالتعويض في المعادلة (24.4) نجد

$$g(n) = \Theta\left(\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right) . \qquad (24.4)$$

نحدّ هذا المجموع بالتدوين– Θ كما في الحالة 1، ولكن لا نحصل هذه المرّة على سلسلة هندسية، بل نكتشف هنا أن كل حدود المجموع متماثلة:

$$\begin{aligned}\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} &= n^{\log_b a} \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^{\log_b a}}\right)^j \\ &= n^{\log_b a} \sum_{j=0}^{\log_b n-1} 1 \\ &= n^{\log_b a} \log_b n .\end{aligned}$$

وبتعويض هذه العبارة بالمجموع في المعادلة (24.4) نجد

$$\begin{aligned}g(n) &= \Theta(n^{\log_b a} \log_b n) \\ &= \Theta(n^{\log_b a} \lg n) ,\end{aligned}$$

وبهذا نكون قد برهنا الحالة 2.

تُبرهَن الحالة 3 بطريقة مماثلة. فلما كان $f(n)$ يظهر في التعريف (22.4) للدالة $g(n)$، ولما كانت كل حدود الدالة $g(n)$ موجبة، يمكننا استنتاج أن $g(n) = \Omega(f(n))$ لقوى b الصحيحة. نفترض في نص التوطئة أن $af(n/b) \le cf(n)$ حيث $c < 1$ ثابت ما، وقيم n كلها كبيرة كفاية. نعيد كتابة هذه الفرضية كما يلي $f(n/b) \le (c/a)f(n)$، ونكرّر ذلك j مرة، فنجد $f(n/b^j) \le (c/a)^j f(n)$، أو ما يكافئه $a^j f(n/b^j) \le c^j f(n)$، حيث نفترض أن قيم n في عمليات التكرار كبيرة كفاية. تحقق جميعُ الحدود ما عدا عدد ثابت منها هذه المتراجحة، ويكون أصغر الحدود التي لا يحققها n/b^{j-1}، والذي يتحقق من أجله أن $a^j f(n/b^{j-1}) = O(1)$.

وبالتعويض في المعادلة (22.4) وبالتبسيط نحصل على سلسلة هندسية، ولكن حدود هذه السلسلة متناقصة، خلافًا للحالة 1. نستخدم الحد $O(1)$ لنختزل فيه كل الحدود التي لا تحقق فرضيتنا بأن n كبيرة كفاية:

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$$

$$\leq \sum_{j=0}^{\log_b n-1} c^j f(n) + O(1)$$
$$\leq f(n)\sum_{j=0}^{\infty} c^j + O(1)$$
$$= f(n)\left(\frac{1}{1-c}\right) + O(1)$$
$$= O(f(n)) \ ,$$

وذلك لأن c ثابت. وهكذا يمكننا أن نستنتج أن $g(n) = \Theta(f(n))$ لقوى b الصحيحة. وبإتمام برهان الحالة 3، يكون قد تمّ برهان التوطئة. ■

يمكننا الآن برهان نسخة من المبرهنة الرئيسة في الحالة التي تكون فيها n قوّة صحيحة لـ b.

توطئة 4.4

ليكن $a \geq 1$ و $b > 1$ ثابتين، ولتكن $f(n)$ دالة موجبة معرَّفة على القوى الصحيحة لـ b. نعرّف $T(n)$ على القوى الصحيحة لـ b بالعلاقة العَوْدية التالية:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \ , \\ aT(n/b) + f(n) & \text{if } n = b^i \ , \end{cases}$$

حيث i عدد طبيعي. وعندها يمكن حدّ $T(n)$ بالمقاربة عند القوى الصحيحة لـ b كالتالي:

1. إذا كان، $f(n) = O(n^{\log_b a-\epsilon})$ حيث $\epsilon > 0$ ثابت ما، فإن $T(n) = \Theta(n^{\log_b a})$.
2. إذا كان $f(n) = \Theta(n^{\log_b a})$، فإن $T(n) = \Theta(n^{\log_b a}\lg n)$.
3. إذا كان $f(n) = \Omega(n^{\log_b a+\epsilon})$ حيث $\epsilon > 0$ ثابت ما، وإذا كان $af(n/b) \leq cf(n)$ حيث $c < 1$ ثابت ما، و n كبيرة كفاية، فإن $T(n) = \Theta(f(n))$.

البرهان نستخدم الحدود في التوطئة 3.4 لنقدّر المجموع (21.4) من التوطئة 2.4. في الحالة 1 نجد

$$T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a})$$
$$= \Theta(n^{\log_b a}) \ ,$$

وفي الحالة 2،

$$T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a}\lg n)$$
$$= \Theta(n^{\log_b a}\lg n) \ .$$

وفي الحالة 3،

$$T(n) = \Theta(n^{\log_b a}) + \Theta(f(n))$$
$$= \Theta(f(n)) ,$$

لأن $f(n) = \Omega(n^{\log_b a+\epsilon})$. ■

2.6.4 الأرضيات والأسقف

لإتمام برهان المبرهنة الرئيسة، علينا أن نوسّع تحليلنا ليشمل الحالة التي تكون فيها الأرضيات والأسقف floors and ceilings مستخدمةً في العلاقة العَوْدية الرئيسة، بحيث تكون العلاقة العَوْدية معرّفة على كل الأعداد الصحيحة، وليس على القوى الصحيحة لـ b فقط. من السهل الحصول على حدٍّ أدنى للدالة

$$T(n) = aT(\lceil n/b\rceil) + f(n) \tag{25.4}$$

وعلى حدٍّ أعلى للدالة

$$T(n) = aT(\lfloor n/b\rfloor) + f(n) \tag{26.4}$$

إذ يمكن دفع المتراجحة $\lceil n/b\rceil \geq n/b$ إلى الحالة الأولى للحصول على النتيجة المطلوبة، ويمكن دفع المتراجحة $\lfloor n/b\rfloor \leq n/b$ إلى الحالة الثانية. إن إيجاد حدٍّ أدنى للعلاقة العَوْدية (26.4) يتطلب استخدام الطريقة نفسها لإيجاد حدٍّ أعلى للعلاقة العَوْدية (25.4)، ولذلك، فإننا سنكتفي بتقديم حدٍّ للأخيرة.

نغيّر شجرة العَوْدية في الشكل 7.4 لنولّد شجرة العَوْدية في الشكل 8.4. ومع نزولنا في شجرة العَوْدية، نحصل على سلسلة من الاستدعاءات العَوْدية على المحدِّدات.

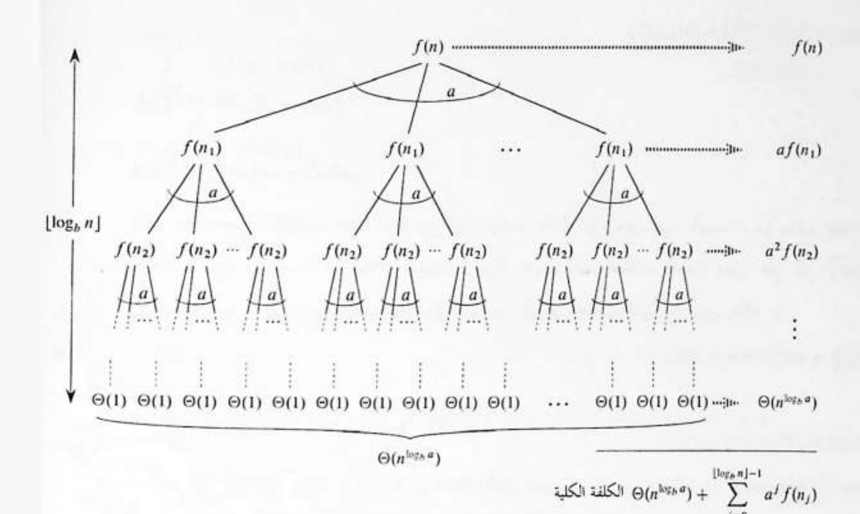
$$n ,$$
$$\lceil n/b\rceil ,$$
$$\lceil\lceil n/b\rceil/b\rceil ,$$
$$\lceil\lceil\lceil n/b\rceil/b\rceil/b\rceil ,$$
$$\vdots$$

لنسمِّ n_j العنصر ذا الرقم j في المتتالية، حيث

$$n_j = \begin{cases} n & \text{if } j = 0 , \\ \lceil n_{j-1}/b\rceil & \text{if } j > 0 . \end{cases} \tag{27.4}$$

غرضنا الأول هو أن نحدّد العمق k بحيث يكون n_k ثابتًا، باستخدام المتراجحة $\lceil x\rceil \leq x + 1$، نحصل على

$$n_0 \leq n ,$$
$$n_1 \leq \frac{n}{b} + 1 ,$$
$$n_2 \leq \frac{n}{b^2} + \frac{1}{b} + 1 ,$$
$$n_3 \leq \frac{n}{b^3} + \frac{1}{b^2} + \frac{1}{b} + 1 ,$$



الشكل 8.4 شجرة العَوْدية التي تولدّها $T(n) = aT(\lceil n/b \rceil) + f(n)$. المحدّد العَوْدي n_j معطى في المعادلة (27.4).

وعمومًا، لدينا

$$
\begin{aligned}
n_j &\le \frac{n}{b^j} + \sum_{i=0}^{j-1} \frac{1}{b^i} \\
&< \frac{n}{b^j} + \sum_{i=0}^{\infty} \frac{1}{b^i} \\
&= \frac{n}{b^j} + \frac{b}{b-1} \, .
\end{aligned}
$$

وإذا جعلنا $j = \lfloor \log_b n \rfloor$ نحصل على

$$
\begin{aligned}
n_{\lfloor \log_b n \rfloor} &< \frac{n}{b^{\lfloor \log_b n \rfloor}} + \frac{b}{b-1} \\
&< \frac{n}{b^{\log_b n - 1}} + \frac{b}{b-1} \\
&= \frac{n}{n/b} + \frac{b}{b-1} \\
&= b + \frac{b}{b-1} \\
&= O(1) \, ,
\end{aligned}
$$

وهكذا نرى أنه عند العمق $\lfloor \log_b n \rfloor$، يكون حجم المسألة على الأكثر ثابتًا.

ومن الشكل 8.4 نرى أنّ

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\lfloor \log_b n \rfloor - 1} a^j f(n_j) \ , \tag{28.4}$$

وهي علاقة مشابهة كثيرًا للمعادلة (21.4)، باستثناء كون n عددًا صحيحًا اختياريًّا، ولا ينحصر في مجموعة القوى الصحيحة لـ b.

يمكننا الآن حساب قيمة المجموع

$$g(n) = \sum_{j=0}^{\lfloor \log_b n \rfloor - 1} a^j f(n_j) \tag{29.4}$$

من المعادلة (28.4)، وبطريقة مشابهة لبرهان التوطئة 3.4. نبدأ بالحالة 3: إذا كان $af(\lceil n/b \rceil) \leq cf(n)$ عندما $n > b + b/(b-1)$، حيث $c < 1$ ثابت، فهذا يستتبع أن يكون $a^j f(n_j) \leq c^j f(n)$. ولهذا السبب يمكن حساب المجموع في المعادلة (29.4) تمامًا كما في التوطئة 3.4. وفي الحالة 2، لدينا $f(n) = \Theta(n^{\log_b a})$. إذا استطعنا أن نبيّن أن $f(n_j) = O(n^{\log_b a}/a^j) = O((n/b^j)^{\log_b a})$، كان بمقدورنا إتمام البرهان في الحالة 2 من التوطئة 3.4. لاحظ أن $j \leq \lfloor \log_b n \rfloor$ يقتضي $b^j/n \leq 1$. إن الحدّ $f(n) = O(n^{\log_b a})$ يقتضي وجود ثابت $c > 0$ بحيث يكون، مهما كان n_j كبيرًا كفاية،

$$\begin{aligned}
f(n_j) &\leq c\left(\frac{n}{b^j} + \frac{b}{b-1}\right)^{\log_b a} \\
&= c\left(\frac{n}{b^j}\left(1 + \frac{b^j}{n} \cdot \frac{b}{b-1}\right)\right)^{\log_b a} \\
&= c\left(\frac{n^{\log_b a}}{a^j}\right)\left(1 + \left(\frac{b^j}{n} \cdot \frac{b}{b-1}\right)\right)^{\log_b a} \\
&\leq c\left(\frac{n^{\log_b a}}{a^j}\right)\left(1 + \frac{b}{b-1}\right)^{\log_b a} \\
&= O\left(\frac{n^{\log_b a}}{a^j}\right) ,
\end{aligned}$$

لأن $c(1 + b/(b-1))^{\log_b a}$ ثابت. وهكذا نكون قد برهنا الحالة 2. وأما برهان الحالة 1، فمماثل له تقريبًا. والفكرة الأساسية هي أن نبرهن الحدّ $f(n_j) = O((n/b^j)^{\log_b a - \epsilon})$، وهذا مماثل للبرهان المقابل في الحالة 2، إلا أن العمليات الجبرية أكثر دقة.

لقد برهنا الآن الحدود العليا في المبرهنة الرئيسة على كل الأعداد الصحيحة n، أما برهان الحدود الدنيا فمشابه لما سبق.

تمارين

★ *1-6.4*

أعطِ عبارة بسيطة ودقيقة لـ n_j في المعادلة (27.4) في الحالة التي يكون فيها b عددًا صحيحًا موجبًا بدلاً من أن يكون عددًا حقيقيًّا اختياريًّا.

★ *2-6.4*

بيّن أنه إذا كان $f(n) = \Theta(n^{\log_b a} \lg^k n)$، حيث $k \geq 0$، فإن حل العلاقة العَوْدية العامة هو $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$. وللتبسيط، اقتصر في تحليلك على القوى الصحيحة لـ b.

★ *3-6.4*

بيّن أن في الحالة 3 من المبرهنة الرئيسة تكرارًا في الشروط، إذ إن شرط الانتظام $af(n/b) \leq cf(n)$ حيث $c < 1$ ثابت ما، يقتضي وجود ثابت $\epsilon > 0$ بحيث يكون $f(n) = \Omega(n^{\log_b a+\epsilon})$.

مسائل

1-4 أمثلة على العلاقات العَوْدية

أعطِ حدودًا عليا ودنيا لـ $T(n)$ لكلٍّ من العلاقات العَوْدية التالية. افترض أن $T(n)$ ثابت عندما $n \leq 2$. اجعل حدودك ملاصقة قدر الإمكان، وعلّل أجوبتك.

أ. $T(n) = 2T(n/2) + n^4$.

ب. $T(n) = T(7n/10) + n$.

ت. $T(n) = 16T(n/4) + n^2$.

ث. $T(n) = 7T(n/3) + n^2$.

ج. $T(n) = 7T(n/2) + n^2$.

ح. $T(n) = 2T(n/4) + \sqrt{n}$.

خ. $T(n) = T(n-2) + n^2$.

2-4 تكاليف تمرير الموسطات

نفترض في هذا الكتاب أن تمرير الموسطات في استدعاءات الإجرائيات يستغرق زمنًا ثابتًا، ولو كنا نمرِّر صفيفةً من N عنصرًا. إن هذه الفرضيّة صحيحة في معظم الأنظمة لأن ما يُمرَّر هو مؤشر على هذه الصفيفة، وليس الصفيفة نفسها. تدرس هذه المسألة ملابسات استخدام ثلاث استراتيجيات لتمرير الموسطات.

. صفيفة ممرّرة باستخدام مؤشر. الزمن هو: $\text{Time} = \Theta(1)$.

. صفيفة ممرّرة بالنسخ. الزمن هو: $\text{Time} = \Theta(N)$، حيث N حجم الصفيفة.

. صفيفة ممرّرة بنسخ الجزء الذي قد تنفذ إليه الإجرائية المستدعاة فقط. الزمن هو: $\text{Time} = \Theta(q - p + 1)$ إذا كانت الصفيفة الجزئية الممررة هي $A[p\,..\,q]$.

أ. لنأخذ خوارزمية البحث الثنائي العَوْدية للعثور على عدد في صفيفة مرتبة (انظر التمرين 2.3-5). أعط العلاقات العَوْدية لأزمنة تنفيذ البحث الثنائي في أسوأ الحالات عندما تمرَّر الصفيفات باستخدام كل طريقة من الطرق السابقة، وأعطِ حدودًا عليا جيّدة لحلول العلاقات العَوْدية. افترض أن N حجم المسألة الأصلية، و n حجم المسألة الجزئية.

ب. أعِدْ الجزء (أ) على خوارزمية MERGE-SORT من المقطع 2.3.1.

3-4 أمثلة إضافية على العلاقات العَوْدية

عطِ حدودًا عليا ودنيا مقاربة لـ $T(n)$ لكل من العلاقات العَوْدية التالية. افترض أن $T(n)$ ثابت عندما تكون n صغيرة كفاية. اجعل حدودك ملاصقة قدر الإمكان، وعلِّل أجوبتك

أ. $T(n) = 4T(n/3) + n \lg n$.

ب. $T(n) = 3T(n/3) + n/\lg n$.

ت. $T(n) = 4T(n/2) + n^2\sqrt{n}$.

ث. $T(n) = 3T(n/3 - 2) + n/2$.

ج. $T(n) = 2T(n/2) + n/\lg n$.

ح. $T(n) = T(n/2) + T(n/4) + T(n/8) + n$.

خ. $T(n) = T(n-1) + 1/n$.

د. $T(n) = T(n-1) + \lg n$.

ذ. $T(n) = T(n-2) + 1/\lg n$.

ر. $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

4-4 أعداد فيبوناتشي

تقدِّم هذه المسألة خصائص جديدة لأعداد فيبوناتشي المعرّفة بالعلاقة العَوْدية (3.22). سوف نستخدم طريقة

الدوال المولِّدة لحل علاقة فيبوناتشي العَوْدية. نعرّف $\mathcal{F}$ ***الدالة المولِّدة generating function*** (أو ***سلسلة القوى الصوريّة formal power series***) كالتالي:

$$\mathcal{F}(z) = \sum_{i=0}^{\infty} F_i z^i$$

$$\mathcal{F}(z) = 0 + z + z^2 + 2z^3 + 3z^4 + 5z^5 + 8z^6 + 13z^7 + 21z^8 + \cdots ,$$

حيث F_i هو عدد فيبوناتشي ذو الترتيب i.

أ. بيّن أنّ $\mathcal{F}(z) = z + z\mathcal{F}(z) + z^2\mathcal{F}(z)$

ب. بيّن أن

$$\begin{aligned}\mathcal{F}(z) &= \frac{z}{1-z-z^2} \\ &= \frac{z}{(1-\phi z)(1-\hat{\phi} z)} \\ &= \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z} - \frac{1}{1-\hat{\phi} z}\right),\end{aligned}$$

حيث

$$\phi = \frac{1+\sqrt{5}}{2} = 1.61803 \ldots$$

و

$$\hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803 \ldots .$$

ت. برهن أن

$$\mathcal{F}(z) = \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i) z^i .$$

ث. استخدم الجزء (ت) لتبرهن أن $F_i = \phi^i/\sqrt{5}$ عندما $i > 0$، مدوّرة إلى أقرب عدد صحيح. (*تلميح*: لاحظ أن $|\hat{\phi}| < 1$.)

5-4 *اختبار رقاقات VLSI*

لدى الأستاذ ديوجينس n رقاقة VLSI[1] من المفترض أن تكون متماثلة: والقادرة من حيث المبدأ على أن تُستخدم لاختبار بعضها بعضًا. تَستخدِم لوحة اختبار الأستاذ رقاقتين في الوقت نفسه. عندما تكون اللوحة

---

[1] تعني VLSI "very large scale integration"، وهي تقانة رقاقات الدارات المتكاملة المستخدمة في تصنيع معظم المعالجات الصغرّية اليوم.

محمّلة، تختبر كلُّ رقاقةٍ الرقاقةَ الأخرى وتبيِّن حالتَها (جيدة أو سيئة). فإذا كانت الرقاقة جيدة، فإنها تعطي بيانًا صحيحًا دائمًا عن حالة الرقاقة الأخرى، غير أن الأستاذ لا يمكن أن يثق بجواب شريحة سيئة. لذا، فإن نتائج الاختبار الأربعة الممكنة هي كالتالي:

بيان الرقاقة A	بيان الرقاقة B	النتيجة
B في حالة جيدة	A في حالة جيدة	كلتاهما في حالة جيدة، أو كلتاهما في حالة سيئة
B في حالة جيدة	A في حالة سيئة	واحدة على الأقل في حالة سيئة
B في حالة سيئة	A في حالة جيدة	واحدة على الأقل في حالة سيئة
B في حالة سيئة	A في حالة سيئة	واحدة على الأقل في حالة سيئة

أ. بيّن أنه إذا كانت $n/2$ رقاقة على الأقل في حالة سيئة، فإن الأستاذ لا يستطيع تحديد الرقاقات الجيّدة مهما كانت الاستراتيجية التي يعتمدها باستخدام هذا النوع من الاختبارات الثنائية. افترضْ أن الرقاقات السيئة تتواطأ لتخدع الأستاذ.

ب. لنتأمل في مسألة العثور على رقاقة جيّدة واحدة من بين n رقاقة، بافتراض وجود أكثر من $n/2$ رقاقة جيدة. بيّن أن $\lfloor n/2 \rfloor$ اختبارًا ثنائيًا كافٍ لاختزال المسألة إلى مسألة أخرى بنصف الحجم تقريبًا.

ت. بيّن أنه يمكن معرفة الرقاقات الجيّدة بـ $\Theta(n)$ اختبارًا ثنائيًا، بافتراض وجود أكثر من $n/2$ رقاقة جيدة. أعطِ العلاقة العَوْدية التي تصف عدد الاختبارات ثم حُلَّها.

6-4 صفيفات مونج

نقول عن صفيفة A مؤلفة من $m \times n$ عددًا حقيقيًّا إنها ***صفيفة مونج Monge Array*** إذا تحقّق

$$A[i,j] + A[k,l] \leq A[i,l] + A[k,j] .$$

أيًّا كان i و j و k و l حيث $1 \leq i < k \leq m$ و $1 \leq j < l \leq n$.

وبعبارة أخرى، عندما نختار صفين وعمودين من صفيفة مونج، وننظر إلى العناصر الأربعة على تقاطع السطرين والعمودين، يكون مجموع العنصرين الأعلى الأيسر والأدنى الأيمن أصغر أو يساوي مجموع العنصرين الأدنى الأيسر والأعلى الأيمن. مثلاً، الصفيفة التالية هي صفيفة مونج:

10	17	13	28	23
17	22	16	29	23
24	28	22	34	24
11	13	6	17	7
45	44	32	37	23
36	33	19	21	6
75	66	51	53	34

أ. برهن أن صفيفة ما هي صفيفة مونج إذا وفقط إذا كان لدينا

$A[i,j] + A[i+1,j+1] \leq A[i,j+1] + A[i+1,j]$.

أيًّا كان $i = 1,2,\dots,m-1$ و $j = 1,2,\dots,n-1$.

(تلميح: فيما يتعلق بالجزء "إذا"، استخدم الاستقراء على الأسطر والأعمدة على نحو منفصل.)

ب. الصفيفة التالية ليست صفيفة مونج. غيِّر عنصرًا واحدًا لجعلها صفيفة مونج. (تلميح: استخدم الجزء أ.)

37	23	22	32
21	6	7	10
53	34	30	31
32	13	9	6
43	21	15	8

ت. ليكن $f(i)$ مؤشر العمود الذي يحوي أصغر عنصر إلى أقصى اليسار في السطر i. برهن أن $f(1) \leq f(2) \leq \dots \leq f(m)$ أيًّا كان بُعدا $m \times n$ صفيفة مونج.

ث. لدينا هنا وصف لخوارزمية فرّقْ-تسدْ تَحسب أصغر عنصر إلى أقصى اليسار في كل سطر من صفيفة مونج A بُعداها $m \times n$:

ابنِ مصفوفة جزئية A' من A، تتألف من الأسطر الزوجية من A. حدّد عَوْديًّا أصغر عنصر إلى أقصى اليسار في كل سطر من A'. ثمّ احسب أصغر عنصر إلى أقصى اليسار في الأسطر المفردة من A.

اشرح كيف يمكن حساب أصغر عنصر إلى أقصى اليسار في الأسطر المفردة من A في زمن $O(m+n)$. (بافتراض أننا نعرف أصغر عنصر إلى اليسار في الأسطر الزوجيّة.)

ج. اكتب العلاقة العَوْدية التي تصف زمن تنفيذ الخوارزمية الموصَّفة في الجزء (ث). بيّن أن حلّها هو $O(m + n\log m)$.

ملاحظات الفصل

تَعود بدايات طريقة فرّق-تسد لتصميم الخوارزميات إلى ما قبل 1962 في مقال لـ Karatsuba و Ofman [194]. إلا أنها – تبعًا لـ Heideman و Johnson و Burras [163]– قد تكون استخدمت قبل ذلك بكثير، فقد صمم C. F. Gauss أول خوارزمية تحويل فورييه السريع في 1805، حيث تجزِّئ صياغة Gauss المسألةَ إلى مسائل جزئية أصغر لتركّب حلولها معًا.

إن مسألة الصفيفة الجزئية الصغرى المذكورة في المقطع 1.4 ما هي إلاّ تعديلٌ طفيف على مسألة درسها Bentley [43] في الفصل السابع.

لقد أَحدثت خوارزميةُ شتراسن Strassen [325] الكثيرَ من الإثارة عندما نُشرت في عام 1969؛ إذ إن قلّة قليلة فقط هم الذين كانوا يتخيلون إمكانية إيجاد خوارزمية أسرع بالمقارنة من إجراء SQUARE-MATRIX-MULTIPLY الأساسي. ولقد أُجريت تحسينات على الحدّ الأعلى لجداء المصفوفات منذ ذلك الحين. إن أكثر الخوارزميات فعالية بالمقارنة لحساب جداء مصفوفتين $n \times n$ حتى يومنا هذا هي الخوارزمية المنسوبة إلى Coppersmith و Winograd [79]، وهي تحقق زمن تنفيذ $O(n^{2.376})$. أما أفضل حد أدنى معروف، فهو الحد البديهي $\Omega(n^2)$ (هو بديهي لأنه يجب ملء n^2 عنصرًا في مصفوفة الجداء).

أما من وجهة النظر العملية، فإن خوارزمية شتراسن ليست على الأغلب الطريقة المفضلة لحساب جداء المصفوفات، للأسباب الأربعة التالية:

1. العامل الثابت المخفي في $\Theta(n^{\lg 7})$ لزمن تنفيذ خوارزمية شتراسن أكبر من العامل الثابت الموجود في $\Theta(n^3)$ لزمن تنفيذ إجراء SQUARE-MATRIX-MULTIPLY.
2. عندما تكون المصفوفات متخلخلة sparse، فإن الطرق المصممة لهذا النوع من المصفوفات أسرع.
3. لا تتمتع خوارزمية شتراسن بالاستقرار الرقمي نفسه لخوارزمية SQUARE-MATRIX-MULTIPLY. وبعبارة أخرى، إن الدقة المحدودة للعمليات الحسابية المحوسبة على القيم غير الصحيحة تتسبب في تراكم أخطاء أكبر في خوارزمية شتراسن منها في خوارزمية SQUARE-MATRIX-MULTIPLY.
4. تستهلك المصفوفات الجزئية المتكوِّنة في مستويات الاستدعاءات العَوْدية الذاكرة.

في عام 1990 تقريبًا، خفَّفت الأبحاث من عمق أثر السببين الأخيرين. فقد برهن Higham [167] أن الفرق في الاستقرار الرقمي كان مبالغًا فيه؛ فعلى الرغم من أن خوارزمية شتراسن غير مستقرة رقميًا كفاية في بعض التطبيقات، إلا أنها في الحدود المقبولة في تطبيقات أخرى. يناقش Bailey و Lee و Simon [32] تقنياتٍ للحد من متطلبات الذاكرة في خوارزمية شتراسن.

عمليًّا، تَستخدم التنجيزات السريعة لجداء المصفوفات الكثيفة خوارزميةَ شتراسن للمصفوفات التي تتجاوز أحجامها "نقطة تجاوز"، وتتحول إلى طريقة أبسط عندما يقل حجم المسألة الجزئية عن نقطة التجاوز هذه. إن القيمة الدقيقة لنقطة التجاوز تتعلق كثيرًا بالنظام الحاسوبي. وقد أعطت الدراسات التي تجاهلت تأثير الذاكرة السريعة cache والنقل عبر أنابيب pipelining نقاط تجاوز منخفضة حتى $n = 8$ (تبعًا لـ Higham [167]) أو $n = 12$ (تبعًا لـ Huss-Lederman وآخرون [186]). فيما طوّر D'Alberto و Nicolau [82] منهجًا تكيفيًّا لتحديد نقطة التجاوز، وذلك بتطبيق تجارب معيارية benchmarking على الحزمة البرمجية عند تنصيبها. ووجدا نقاط تجاوزٍ على أنظمة متنوعة تقع ما بين $n = 400$ و $n = 2150$، ولم يتمكنا من إيجاد نقطة تجاوز على نظامٍ أو نظامين.

بدأت دراسة العلاقات العَوْدية باكرًا في العام 1202 على يد فيبوناتشي L. Fibonacci، الذي سُميت أعداد فيبوناتشي باسمه. وأدخل دوموافر A. De Moivre طريقة الدوال المولّدة لحلّ العلاقات العَوْدية (انظر المسألة 4-4). الطريقة الرئيسة مستقاة من Bentley و Haken و Saxe [44]، التي تقدّم طريقة موسّعة مبررة بالتمرين 4.6-2. يبيّن كل من Knuth [209] و Liu [237] كيف يمكن حل العلاقات العَوْديّة الخطيّة باستخدام طريقة الدوال المولّدة. ويضمّ كل من Purdom و Brown [287] و Graham و Knuth و Patashnik [152] نقاشًا موسّعًا حول حل العلاقات العَوْديّة.

قدّم العديد من الباحثين، ومنهم Akra و Bazzi [13]، و Roura [299] و Verma [346] و Yap [360] طرقًا لحلّ علاقات عَوْديّة من نمط فرّق-تسد أعمّ من تلك التي يمكن حلّها باستخدام الطريقة الرئيسة. ونشرح هنا نتيجة أعمال Akra و Bazzi التي عدّلها Leighton [228]، والتي تنطبق على علاقات عَوْدية من الشكل

$$T(x) = \begin{cases} \Theta(1) & \text{if } 1 \le x \le x_0 \ , \\ \sum_{i=1}^{k} a_i T(b_i x) + f(x) & \text{if } x > x_0 \ , \end{cases} \tag{30.4}$$

حيث:

- $x \ge 1$ عدد حقيقي؛
- x_0 عدد ثابت بحيث يكون $x_0 \ge 1/b_i$ و $x_0 \ge 1/(1 - b_i)$ حيث $i = 1, 2, \dots, k$،
- a_i ثابت موجب حيث $i = 1, 2, \dots, k$
- b_i ثابت في المجال $0 < b_i < 1$ حيث $i = 1, 2, \dots, k$
- $k \ge 1$ عدد صحيح ثابت،
- $f(x)$ دالة موجبة تحقق شرط ***النمو الحدودي Polynomial-growth condition***: يوجد ثابتان موجبان c_1 و c_2 حيث، مهما كان $x \ge 1$، و $i = 1, 2, \dots, k$، ومهما كان u حيث $b_i x \le u \le x$، يكون لدينا $c_1 f(x) \le f(u) \le c_2 f(x)$. (إذا كان $|f'(x)|$ محدودًا من الأعلى بكثير حدود ما لـ x، فإن الدالة $f(x)$ تحقق شرط النمو الحدودي. على سبيل المثال تحقق الدالة $f(x) = x^{\alpha} \lg^{\beta} x$ هذا الشرط مهما كان الثابتان الحقيقيان α و β.)

وعلى الرغم من أنه لا يمكن تطبيق الطريقة الرئيسة على علاقات عَوْديّة مثل $T(n) = T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + O(n)$، فإنه يمكن تطبيق طريقة Akra-Bazzi. ولحل العلاقة العَوْدية (30.4)، فإننا نوجد أولاً العدد الحقيقي الوحيد p بحيث يكون $\sum_{i=1}^{k} a_i b_i^{p} = 1$. (إن p موجود دائمًا) وعندها يكون حل العلاقة العَوْدية هو

$$T(n) = \Theta\left(x^p\left(1 + \int_1^x \frac{f(u)}{u^{p+1}} du\right)\right).$$

قد يكون من الصعوبة بمكان استخدام طريقة Akra-Bazzi، ولكنها تفيد في حل علاقات عَوْديّة تنمذج تقسيم المسألة إلى مسائل جزئية مختلفة الحجم جوهريًّا. أما الطريقة الرئيسة، فأسهل في الاستخدام، إلا أنه لا يمكن تطبيقها إلا عندما تكون أحجام المسائل الجزئية متساوية.

5 التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة

يعرّف هذا الفصل بالتحليل الاحتمالي والخوارزميات ذات العشوائية المضافة probabilistic analysis and randomized algorithms. فإذا لم تكن على معرفة بأسس نظرية الاحتمالات، فينبغي أن تقرأ الملحق–ت، الذي يستعرض هذه المادة. وسنعود في هذا الكتاب عدّة مرات إلى التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة.

1.5 مسألة التوظيف

افترضْ أنك تريد أن توظّف مساعدًا جديدًا في المكتب، وقد أخفقت محاولاتك السابقة للتوظيف، فقررت أن تلجأ إلى وكالة توظيف. سترسل لك وكالةُ التوظيف مرشحًا كل يوم، فتقابل ذلك الشخص ثمّ تقرّر توظيفه أو عدم توظيفه. وعليك أن تسدد لوكالة التوظيف عمولة صغيرة مقابل إجراء مقابلة مع متقدّم ما. لكن سيكون توظيف متقدّم ما أكثر كلفة إذ عليك أن تستغني عن مساعدك الحالي وأن تسدد عمولة كبيرة إلى وكالة التوظيف. وتضمن لك الوكالة أن تقدم لك على الدوام الشخصَ الأفضل للمهمة المتاحَ لديها. ولهذا السبب قررتَ أنه – بعد مقابلة كل متقدّم – إذا كان هذا المتقدّم أفضل من مساعدك الحالي، فإنك ستستغني عن المساعد الحالي وتوظّف المتقدّم الجديد. لا مانع لديك من تسديد ثمن هذه الاستراتيجية، إلا أنك ترغب في تقدير هذا الثمن.

يعبّر الإجراء HIRE-ASSISTANT، المعطى هنا، عن استراتيجيّة التوظيف هذه بشبه رماز. ونفترض أن المرشحين لوظيفة مساعد المكتب مرقّمون من 1 إلى n. يفترض الإجراء أنك قادر، بعد مقابلة المرشح i على تحديد كون هذا المرشح i هو أفضل مرشح قابلته حتى ذلك الوقت. ينشئ الإجراء، بهدف التهيئة، مرشحًا خلبيًّا، رقمه 0 هو الأسوأ من بين كل المرشحين الآخرين.

```
HIRE-ASSISTANT(n)
1  best = 0          // candidate 0 is a least-qualified dummy candidate
2  for i = 1 to n
3       interview candidate i
```

```
4        if candidate i is better than candidate best
5            best = i
6            hire candidate i
```

يختلف نموذج الكلفة لهذه المسألة عن النموذج المشروح في الفصل 2. نحن لا نهتم هنا بزمن تنفيذ HIRE-ASSISTANT، بل بالكلفة الناتجة عن المقابلات والتوظيف. قد يبدو ظاهريًا أن تحليل كلفة هذه الخوارزمية مختلفٌ تمامًا عن تحليل زمن تنفيذ الفرز بالدمج مثلاً، إلا أن طرق التحليل المستخدمة هي نفسها سواءً أكنّا نحلّل كلفة أم زمن تنفيذ. ففي كلتا الحالتين، نحن نعدّ عدد مرات تنفيذ عمليات أساسيّة معيّنة.

للمقابلة كلفة منخفضة، ولتكن c_i، على حين أن للتوظيف كلفة مرتفعة، ولتكن c_h. وليكن m عدد الأشخاص الموظّفين. عندها تكون الكلفة الكليّة المقابلة لهذه الخوارزميات هي $O(c_in + c_hm)$. ومهما كان عدد الأشخاص الذين نوظفهم، فإننا نقابل دائمًا n مرشحًا، وهكذا فإن كلفة المقابلات c_in ستترتب دائمًا علينا. ولذلك فإننا سنركز على تحليل c_hm، كلفة التوظيف. فهذه القيمة تتغير مع كل تنفيذ للخوارزمية.

يفيد هذا السيناريو بوصفه نموذجًا لمنهجية عمل محوسب سائدة. فكثيرًا ما نحتاج إلى إيجاد القيمة العظمى أو الصغرى لمتتاليةٍ ما بدراسة كل عنصر فيها والاحتفاظ "بالفائز" الحالي. إن مسألة التوظيف تنمذج مدى تكرار تغيير رؤيتنا للعنصر الفائز حاليًا.

تحليل أسوأ الحالات

نقوم في أسوأ الحالات عمليًا بتوظيف كل مرشح نقابله. ويتحقق ذلك عندما يَرِد المرشحون بالترتيب التصاعدي من حيث الكفاءة، وفي هذه الحالة نوظّف n مرة، بكلفة توظيف كليّة تبلغ $O(c_hn)$.

طبعًا لا يَرِد المرشحون دائمًا بالترتيب التصاعدي من حيث الكفاءة. وفي الحقيقة، لا علم لنا بترتيب ورودهم، وليس لنا أية قدرة على التأثير على هذا الترتيب، ولهذا السبب، من الطبيعي أن نتساءل عمّا نتوقع حدوثه في الحالة الاعتيادية أو الوسطى.

التحليل الاحتمالي

التحليل الاحتمالي probabilistic analysis هو استخدام الاحتمالات في تحليل المسائل، والأكثر شيوعًا هو أن نستخدم التحليل الاحتمالي لتحليل زمن تنفيذ خوارزمية ما، ولكننا نستخدمه أحيانًا لتحليل مقادير أخرى، مثل كلفة التوظيف في إجراء HIRE-ASSISTANT. وللقيام بالتحليل الاحتمالي علينا أن نستخدم معرفتنا بتوزّع المُدْخلات الاحتمالي، أو أن نفترض فرضيات بشأنه. ثمّ نحلّل خوارزميتنا، لحساب زمن التنفيذ المتوقع، ونحسب التوقع على توزيع المُدْخلات الممكنة. وهكذا، فإننا نقوم عمليًا بحساب متوسط زمن التنفيذ على كل المُدْخلات الممكنة. عندما نتحدث عن زمن تنفيذ مماثل لهذا الزمن، فإننا سنشير إليه على أنه ***زمن التنفيذ في الحالة الوسطى average-case running time***.

يجب أن نكون حذرين عند تحديد توزّع المدخلات الاحتمالي. لأنه في بعض المسائل، يكون افتراض بعض الفرضيات بشأن مجموعة المُدْخلات الممكنة منطقيًّا، ويمكننا عندها استخدام التحليل الاحتمالي كطريقة لتصميم خوارزمية فعّالة، وكوسيلة للتعمق في فهم المسألة. أما في مسائل أخرى، فليس بمقدورنا توصيف توزيع معقول للدخل، ولا يمكننا في هذه الحالة استخدام التحليل الاحتمالي.

يمكننا، فيما يخص مسألة التوظيف، افتراض أن المتقدّمين يأتون وفق ترتيب عشوائي. ولكن ماذا يعني ذلك في هذه المسألة؟ نفترض أنه بمقدورنا مقارنة أي مرشحَيْن وتحديد أيٌّ منهما هو الأنسب، أي إن هناك ترتيبًا شاملاً للمرشحين. (انظر تعريف الترتيب الشامل في الملحق ب.) وهكذا يمكننا إعطاء كل مرشح مرتبةً هي عدد وحيد بين 1 و n، وذلك باستخدام $rank(i)$ للإشارة إلى مرتبة المتقدّم i، وباعتماد فرضية أن المرتبة الأعلى تقابل متقدّمًا أكثر كفاءةً. إن القائمة المرتبة $\langle rank(1), rank(2), \ldots, rank(n)\rangle$ هي تبديل على القائمة $\langle 1, 2, \ldots, n\rangle$. إنّ قولنا بأن المتقدّمين يأتون وفق ترتيب عشوائي يكافئ أن نقول إنه من الممكن أن تكون قائمة المراتب هي أي تبديل من تباديل الأعداد من 1 إلى n، والتي يبلغ عددها $n!$، وذلك باحتمال متساوٍ. أو بطريقة أخرى، نقول إن المراتب تشكّل ***تبديلاً عشوائيًّا منتظمًا uniform random permutation***، أي إن كل التباديل الممكنة، وعددها $n!$، ترد باحتمال متساوٍ.

يحتوي المقطع 2.5 تحليلاً احتماليًّا لمسألة التوظيف.

الخوارزميات ذات العشوائية المضافة

نحتاج، لاستخدام التحليل الاحتمالي، إلى بعض المعلومات عن توزّع المُدْخلات. وفي كثير من الأحيان، لا نعرف إلا القليل عن توزّعها. وحتى إن كان لدينا بعض العلم بهذا التوزّع، فقد لا نكون قادرين على نمذجة هذه المعرفة نمذجةً محوسبة. ومع ذلك، يمكننا في كثير من الأحيان، استخدام الاحتمالات والعشوائية أداتَيْن لتصميم الخوارزميات وتحليلها، بأن نجعل سلوك جزء من الخوارزمية عشوائيًّا.

قد يبدو، في مسألة التوظيف، أن المرشحين يُرسَلون إلينا وفق ترتيب عشوائي. ولكن، ليس بمقدورنا أن نعرف إذا كانوا يُرسَلون كذلك فعلاً. ولهذا السبب، يجب أن نزيد في التحكّم في ترتيب مقابلات المرشحين، كي نتمكن من تطوير خوارزمية ذات عشوائية مضافة لمسألة التوظيف. لذا، سنغيّر النموذج قليلاً. سنقول أن لدى وكالة التوظيف n مرشّحًا، وأنها ترسل لنا سلفًا قائمةً بالمرشحين. وفي كل يوم، نحن نختار عشوائيًّا المرشح الذي سنقابله. ومع أننا لا نعرف شيئًا عن المرشحين (ما عدا أسماءَهم)، فقد أجرينا تغييرًا كبيرًا. فبدلاً من أن نعتمد على التخمين بأن المرشحين سيأتون بترتيبٍ عشوائي، زدنا تحكّمنا في العملية، وفرضنا ترتيبًا عشوائيًّا.

عمومًا، نقول عن خوارزمية إنها ***ذات عشوائية مضافة randomized*** إذا كان سلوكها يتحدّد، إضافة إلى الدخل، بالاعتماد على قيم يولدها ***مولّد أعداد عشوائي random-number generator***. سنفترض أن في حوزتنا مولّد أعدادٍ عشوائيًّا RANDOM، وأن الاستدعاء $\text{RANDOM}(a, b)$ يعيد عددًا طبيعيًّا بين a و b،

يمكن أن يكون a أو b، وبحيث تكون كل الأعداد متساوية الاحتمال. فمثلاً، RANDOM(0,1) يعطي 0 باحتمال 1/2 و 1 باحتمال 1/2. ويعيد استدعاء RANDOM(3,7) إحدى القيم 3 أو 4 أو 5 أو 6 أو 7، كل منها باحتمال 1/5. إن كل عدد طبيعي يعيده RANDOM مستقل عن الأعداد المولّدة في استدعاءات سابقة. يمكنك أن تتخيل RANDOM وكأنك ترمي حجر نرد له $(b - a + 1)$ وجهًا لتحصل على خرجه (تقدّم معظم بيئات البرمجة عمليًا ***مولّد أعداد شبه عشوائية pseudorandom-number generator***: وهو خوارزمية حتميّة تعيد أعدادًا "تبدو" إحصائيًا وكأنها عشوائية).

عندما ندرس زمن تنفيذ خوارزمية ذات عشوائية مضافة، نأخذ توقع زمن التنفيذ تبعًا للتوزيع الاحتمالي للقيم التي يعيدها مولد الأعداد العشوائية. نميّز هذه الخوارزميات عن تلك التي يكون فيها الدخل عشوائيًا بأن نشير إلى زمن تنفيذ خوارزمية ذات عشوائية مضافة على أنه ***زمن التنفيذ المتوقع expected running time***. وعمومًا، نتحدث عن زمن التنفيذ في الحالة الوسطى عندما يكون التوزيع الاحتمالي على مدخلات الخوارزمية، ونتحدث عن زمن التنفيذ المتوقع عندما تقوم الخوارزمية نفسها باتخاذ خيارات عشوائية.

تمارين

1-1.5

بيّن أن الفرضيّة التي تقول إننا دائمًا قادرون على تحديد أي مرشح هو الأفضل في السطر 4 من الإجراء HIRE-ASSISTANT تقتضي أن نعرف ترتيبًا شاملاً على مراتب المرشحين.

2-1.5 ★

وصّفْ تنجيزًا للإجراء RANDOM(a, b) تقوم فيه فقط باستدعاءات RANDOM$(0, 1)$. ما هو زمن التنفيذ المتوقع لإجرائيتك باعتبارها دالةً لـ a و b؟

3-1.5 ★

افترض أنك تريد خرجًا يساوي 0 باحتمال 1/2 و 1 باحتمال 1/2. وفي حوزتك إجراء BIASED-RANDOM، خرجه إما 1 أو 0. تخرج 1 باحتمالٍ ما p، و 0 باحتمال $1 - p$، حيث $0 < p < 1$، ولكنك لا تعرف قيمة p. أعط خوارزمية تستخدم BIASED-RANDOM باعتباره إجراءً فرعيًا، وتعيد جوابًا غير منحازٍ، أي تعيد 0 باحتمال 1/2 و 1 باحتمال 1/2. ما هو زمن التنفيذ المتوقع لخوارزميتك بوصفها دالةً لـ p؟

2.5 المتحولات العشوائية المؤشرة

سنستخدم، لتحليل العديد من الخوارزميات، ومنها مسألة التوظيف، متحولات المؤشرات العشوائية التي تمثّل طريقة مناسبة للتحويل بين الاحتمالات والتوقعات. لنفترض أن لدينا فضاءَ عينةٍ S وحدثًا A، فيكون ***المؤشر***

العشوائي [1] ***indicator random variable*** $\mathrm{I}\{A\}$ الموافق للحدث A معرَّفًا كما يلي:

$$\mathrm{I}\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs },\\ 0 & \text{if } A \text{ does not ocuur }. \end{cases} \tag{1.5}$$

[أي يأخذ القيمة 1 إذا وقع الحدث A، و 0 وإذا لم يقع.]

لنحدّد، كمثال بسيط، عدد المرات المتوقع الذي نحصل فيه على "وجه head" عندما نرمي قطعة نقود عادلة. إن فضاء العينة هو $S = \{H, T\}$، مع $\Pr\{H\} = \Pr\{T\} = 1/2$، ويمكننا أن نعرّف هنا المؤشر العشوائي X_H الموافق لأن تأتي الرمية بالنتيجة "وجه"، وهذا ما يعرف بالحدث H. يَعُدّ هذا المتحوّل عدد مرات ظهور "الوجه" في الرمية، فهو يساوي 1 إذا ظهر "الوجه"، و 0 إذا ظهر الجانب الآخر. نكتب:

$$\begin{aligned} X_H &= \mathrm{I}\{H\} \\ &= \begin{cases} 1 & \text{if } H \text{ occurs },\\ 0 & \text{if } T \text{ occurs }. \end{cases} \end{aligned}$$

إن عدد الوجوه المتوقع الحصول عليه في رمية واحدة هو ببساطة القيمة المتوقعة للمؤشّر X_H:

$$\begin{aligned} \mathrm{E}[X_H] &= \mathrm{E}[\mathrm{I}\{H\}] \\ &= 1 \cdot \Pr\{H\} + 0 \cdot \Pr\{T\} \\ &= 1 \cdot (1/2) + 0 \cdot (1/2) \\ &= 1/2 . \end{aligned}$$

إذن، عدد الوجوه المتوقع الحصول عليه في رمية واحدة لقطعة نقد عادلة هو 1/2. وتبيّن التوطئة التالية أن القيمة المتوقعة للمؤشر العشوائي الموافق لحدث A تساوي احتمال وقوع الحدث A.

توطئة 1.5

ليكن لدينا فضاء عينة S وحدث A من فضاء العينة S، وليكن $X_A = \mathrm{I}\{A\}$ ، فيكون $\mathrm{E}[X_A] = \Pr\{A\}$.

البرهان اعتمادًا على تعريف المؤشر العشوائي في المعادلة (1.5)، وعلى تعريف القيمة المتوقعة، لدينا

$$\begin{aligned} \mathrm{E}[X_A] &= \mathrm{E}[\mathrm{I}\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} \\ &= \Pr\{A\} , \end{aligned}$$

حيث تشير $\bar{A}$ إلى $S - A$، متممة A. ■

قد يبدو من المرهق استعمال المؤشرات العشوائية في تطبيق مثل عدّ عدد المرات المتوقع فيه الحصول على

[1] كان من الواجب أن نستخدم التعبير "متحول عشوائي مؤشر" حرفيًا، لكننا ارتأينا – للسهولة – استخدام "مؤشر عشوائي"؛ فمن الواضح أن السياق يدل على أننا نتعامل مع متحولات عشوائية من نمط مؤشر. (المترجم)

وجه عند رمي قطعة نقود واحدة، إلا أنها مفيدة في دراسة حالات نقوم فيها بتجارب عشوائية مكرّرة. فعلى سبيل المثال، تعطينا المؤشرات العشوائية طريقة بسيطة للوصول إلى نتيجة المعادلة (ت.37). ففي هذه المعادلة نحسب عدد الوجوه عند رمي قطعة نقد n مرة، وذلك بأن ندرس على نحو منفصل احتمال الحصول على: 0 وجهًا، وجهٍ واحد، وجهَيْن، إلخ. إن الطريقة البسيطة المقترحة في المعادلة (ت.38) تستخدم ضمنيًّا المؤشرات العشوائية. ولمزيد من الإيضاح، بمقدورنا تسمية X_i المؤشر العشوائي الموافق للحدث المتمثل في الحصول على وجه في الرمية i. أي لدينا: $X_i = \mathrm{I}\{\textbf{the i th flip results in the event } H\}$. ليكن X المتحول العشوائي الذي يمثل عدد الوجوه الكلي في n رمية، بحيث

$$X = \sum_{i=1}^{n} X_i \ .$$

نريد أن نحسب عدد الوجوه المتوقع، لذلك فإننا نأخذ توقع طرفي المعادلة السابقة لنحصل على

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] .$$

تعطي المعادلة السابقة توقع مجموع n متحولاً عشوائيًّا. وبالاعتماد على التوطئة 1.5، يمكننا بسهولة حساب توقع كل متحول عشوائي. وبالاعتماد على (ت.21) – خطيّة التوقع – يصبح حساب توقع المجموع سهلاً: فهو يساوي مجموع توقعات المتحولات العشوائية، وعددها n. إن خطيّة التوقع تجعل من استخدام المؤشرات العشوائية طريقة تحليلية فعّالة؛ ويمكن تطبيقها حتى حين تكون المتحولات العشوائية مرتبطة. بإمكاننا الآن أن نحسب عدد الوجوه المتوقع بسهولة:

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[X_i] \\
&= \sum_{i=1}^{n} 1/2 \\
&= n/2 \ .
\end{aligned}$$

وهكذا، ومقارنةً بالطريقة المستخدمة في المعادلة (ت.37)، نجد أن المؤشرات العشوائية تبسّط الحساب كثيرًا. لذا، فإننا سنستخدم المؤشرات العشوائية في كل هذا الكتاب.

تحليل مسألة التوظيف باستخدام المؤشرات العشوائية

لنعدْ إلى مسألة التوظيف. نريد الآن أن نحسب العدد المتوقع للمرات التي نوظّف فيها موظفًا جديدًا. حتى

نتمكن من استخدام تحليل احتمالي، نفترض أن المرشحين يَصلون وفق ترتيب عشوائي، كما ناقشنا في المقطع السابق. (سنرى في المقطع 3.5 كيف نحذف هذه الفرضية.) ليكن X المتحوّل العشوائي الذي تساوي قيمته عدد مرات توظيف موظف جديد. بإمكاننا أن نطبّق تعريف القيمة المتوقعة من المعادلة (ت.20) فنحصل على

$$\mathrm{E}[X] = \sum_{x=1}^{n} x \Pr\{X = x\} \ ,$$

إلا أن هذا الحساب قد يكون مجهدًا. إذن، سنعمد بدلاً منه إلى استخدام المؤشرات العشوائية التي ستبسّط الحساب كثيرًا.

ولاستخدام المؤشرات العشوائية بدلاً من حساب $\mathrm{E}[X]$ اعتمادًا على متحول واحد موافق لعدد مرات توظيف موظف جديد، نعرّف n متحولاً يتعلّق كلٌّ منها بحقيقة توظيف مرشح محدد أو لا. وبوجهٍ خاص، نعرّف X_i المؤشر العشوائي الموافق للحدث الذي يتمُّ فيه توظيف المرشح i. إذن

$$\begin{aligned} X_i &= \mathrm{I}\{\text{candiadate } i \text{ is hired}\} \\ &= \begin{cases} 1 & \text{if candiadate } i \text{ is hired} \ , \\ 0 & \text{if candiadate } i \text{ is not hired} \ , \end{cases} \end{aligned}$$

[أي إن X_i يأخذ القيمة 1 عندما يُوظَّف المرشح i، و 0 إذا لم يُوظَّف.]

و:

$$X = X_1 + X_2 + \cdots + X_n \ . \tag{2.5}$$

واعتمادًا على التوطئة 1.5، يكون لدينا:

$$\mathrm{E}[X_i] = \Pr\{\text{candiadate } i \text{ is hired}\}$$

[أي $\mathrm{E}[X_i]$ تساوي احتمال توظيف المرشح i.]

علينا إذن حساب احتمال تنفيذ السطرين 5-6 من HIRE-ASSISTANT.

يُوظَّف المرشح i، في السطر 6، عندما يكون i أفضل من كل المرشحين من 1 حتى $i-1$. ولما افترضنا أن المرشحين يَصلون وفق ترتيب عشوائي، فإن أول i مرشحًا ظهروا أيضًا وفق ترتيب عشوائي. واحتمال أن يكون أي واحد من هؤلاء المرشحين i الأوائل هو الأفضل متساوٍ تبعًا لمعلوماتنا. إذن احتمال أن يكون المرشح i أفضل من المرشحين من 1 إلى $i-1$ يساوي $1/i$، وهذا هو أيضًا احتمال توظيفه. واعتمادًا على التوطئة 1.5، نستنتج أن

$$\mathrm{E}[X_i] = 1/i \ . \tag{3.5}$$

بإمكاننا الآن حساب $\mathrm{E}[X]$:

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] && \text{((من المعادلة (2.5))} && (4.5) \\ &= \sum_{i=1}^{n} \mathrm{E}[X_i] && \text{(باستخدام خطية التوقع)} \\ &= \sum_{i=1}^{n} 1/i && \text{((من المعادلة (3.5))} \\ &= \ln n + O(1)\ . && \text{((من المعادلة (أ.7))} && (5.5) \end{aligned}$$

ونحن، وإن كنا نقابل n شخصًا، فإننا في الواقع نوظف وسطيًّا $\ln n$ شخصًا منهم. نلخّص هذه النتيجة في التوطئة التالية.

توطئة 2.5

بافتراض أن المرشحين يَرِدون وفق ترتيب عشوائي، فإن خوارزمية HIRE-ASSISTANT ذات كلفة توظيف كاملة من رتبة $O(c_h \ln n)$.

البرهان ينتج الحدّ مباشرة من تعريفنا لكلفة التوظيف ومن المعادلة (5.5) التي تبين أن عدد مرات التوظيف المتوقع هو تقريبًا $\ln n$. ■

تمثل كلفة التوظيف في الحالة الوسطى تحسّنًا ملموسًا مقارنةً بكلفة التوظيف في أسوأ الحالات $O(c_h n)$.

تمارين

1-2.5

ما احتمال أن توظِّف مرّةً واحدةً فقط في إجراء HIRE-ASSISTANT، بافتراض أن المرشحين يَرِدون وفق ترتيب عشوائي؟ وما احتمال أن توظف n مرّة تمامًا؟

2-2.5

ما احتمال أن توظف مرتين تمامًا في إجراء HIRE-ASSISTANT، بافتراض أن المرشحين يَرِدون وفق ترتيب عشوائي؟

3-2.5

استخدِمِ المؤشرات العشوائية لحساب القيمة المتوقعة لمجموع n زهر نرد.

4-2.5

استخدِمِ المؤشرات العشوائية لحلّ المسألة التالية، التي تُعرَف باسم ***مسألة تعليق القبعات hat-check problem***: في أحد المطاعم يعطي n زبونًا قبعاتهم لموظف كي يعلقها لهم. يعيد هذا الموظف القبعات إلى

الزبائن وفق ترتيب عشوائي. ما هو العدد المتوقع للزبائن الذين يستعيدون قبعاتهم نفسها؟

5-2.5

لتكن $A[1..n]$ صفيفة من n عددًا متمايزاً. إذا كان $i < j$ و $A[i] > A[j]$، نقول عن الزوج (i, j) أنه ***قلْبَة inversion*** لـ A. (انظر المسألة 2-4 لمعرفة المزيد عن القلْبات.) لنفترض أن عناصر A تشكّل تبديلاً عشوائيًا منتظمًا لـ $\langle 1, 2, \ldots, n \rangle$. استخدم المؤشرات العشوائية لحساب عدد القلْبات المتوقع.

3.5 الخوارزميات ذات العشوائية المضافة

بيّنا في المقطع السابق، كيف أن معرفة توزّع المُدْخلات تساعدنا على تحليل سلوك خوارزميةٍ ما في الحالة الوسطى. ولكن في كثير من الأحيان لا نمتلك هذه المعرفة، ومن ثَمَّ لا يمكننا إجراء تحليل للحالة الوسطى. وقد ذكرنا في المقطع 1.5 أنه قد يكون بمقدورنا استخدام خوارزمية ذات عشوائية مضافة.

ففي مسألةٍ كمسألة التوظيف – التي يساعدنا على تحليلها أن نفترض أن جميع التباديل على الدخل متساوية الاحتمال – سيوجّهنا التحليلُ الاحتمالي عند بناء خوارزمية ذات عشوائية مضافة. فبدلاً من افتراض توزيعٍ للمُدْخلات، فإننا نفرض توزيعًا نختاره. وبوجهٍ خاص، قبل تنفيذ الخوارزمية نبدّل المرشحين عشوائيًا بهدف تحقيق خاصية تساوي احتمالات جميع التباديل. ومع أننا غيّرنا الخوارزمية، إلا أننا ما زلنا نتوقع أن نوظف مساعدًا جديدًا في المكتب تقريبًا $\ln n$ مرة، ولكننا نتوقع ذلك الآن *مهما كان* الدخل، بدلاً من أن يكون كذلك بافتراض مُدْخلات مسحوبة تبعًا لتوزيع محدّد.

دعنا نستكشف بعمق أكثر الفرق بين التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة. ذكرنا في المقطع 2.5، أنه بافتراض أن المرشحين يَرِدون تبعًا لترتيب عشوائي، فإن عدد المرات المتوقع الذي نوظّف فيه موظفًا جديدًا هي نحو $\ln n$. لاحظ هنا أن الخوارزمية حتميّة deterministic؛ فعندما نحدد دخلاً ما، سيكون عدد مرات توظيف موظف جديد هو نفسه دائمًا. إضافة إلى ذلك، يختلف عدد مرات توظيف موظف جديد باختلاف المُدْخلات، ويتعلق بمراتب المرشحين. ولمّا كان هذا العدد يتعلق فقط بمراتب المرشحين، يمكننا أن نمثّل أي دخل بأن نذكر مراتب المرشحين بالترتيب، أي $\langle rank(1), rank(2), \ldots, rank(n) \rangle$. فإذا أُعطينا مثلاً قائمة المراتب $A_1 = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle$، يُوظَّف موظف جديد دومًا 10 مرات، إذ إن كلَّ مرشح هو أفضل من سابقه. وسينفَّذ السطران 5-6 في كل تكرار للخوارزمية. وإذا أُعطينا قائمة المراتب $A_2 = \langle 10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1 \rangle$، فسيُوَظَّف موظفٌ جديدٌ مرةً واحدةً فقط، في التكرار الأول. وإذا أُعطينا قائمة المراتب $A_3 = \langle 5, 2, 1, 8, 4, 7, 10, 9, 3, 6 \rangle$، فسيُوظَّف موظفٌ جديدٌ ثلاث مرات، عند مقابلة المرشحين ذوي المراتب 5 و 8 و 10. فإذا تذكّرنا أن كلفة خوارزميتنا تتعلّق بعدد مرات توظيف موظفٍ جديد، فسنجد أن هناك مُدْخلات مُكْلِفة مثل A_1، وأخرى غير مُكلفة مثل A_2، ومُدْخلات ذات

كلفة معتدلة مثل A_3.

لنأخذ من جهة أخرى الخوارزمية ذات العشوائية المضافة التي تقوم أولاً بالتبديل بين المرشحين، ثمّ تحدّد المرشح الأفضل. في هذه الحالة يكمن السلوك العشوائي داخل الخوارزمية وليس في توزيع المُدْخلات. ففي دخل محدَّد، وليكن A_3 السابق، لا نستطيع أن نحدّد عدد المرات التي تُعدَّل فيها القيمة العظمى، لأن هذا العدد يختلف مع كل تنفيذ للخوارزمية؛ فقد يَنتج التبديل A_1 في المرة الأولى التي ننفّذ فيها الخوارزمية على A_3، فتقوم الخوارزمية بـ 10 تعديلات، على حين أن التبديل A_2 قد يَنتج في المرة الثانية التي ننفّذ فيها الخوارزمية، فتقوم بتعديلٍ واحد فقط. وفي المرة الثالثة التي ننفّذها سنقوم بعددٍ من التعديلات الأخرى. ففي كلّ مرةٍ ننفّذ فيها الخوارزمية، يعتمد التنفيذ على الخيارات العشوائية التي قامت بها، والتي من المحتمل أن تختلف عن التنفيذ السابق للخوارزمية. إذن فيما يخص هذه الخوارزمية والعديد من الخوارزميات الأخرى ذات العشوائية المضافة، *ليس هناك دخل محدّد يتسبّب في جعل الخوارزمية تسلك سلوكها في أسوأ الحالات*. حتى إن أسوأ أعدائك غير قادر على توليدِ صفيفةِ دخلٍ سيئة، إذ إن التبديل العشوائي يلغي تأثير ترتيب الدخل. ولا يكون أداء الخوارزمية ذات العشوائية المضافة سيئًا إلا إذا وَلَّد مولِّد الأعداد العشوائية تبديلاً "سيء الحظ".

يتمثل التغيير الوحيد على الرماز فيما يخص مسألة التوظيف بتبديل الصفيفة عشوائيًّا.

```
RANDOMIZED-HIRE-ASSISTANT(n)
1  randomly permute the list of candidates
2  best = 0
3  for i = 1 to n
4       interview candidate i
5       if candidate i is better than candidate best
6           best = i
7           hire candidate i
```

وبهذا التغيير الطفيف، نكون قد بنينا خوارزميةً ذات عشوائية مضافة يشبه أداؤها أداءَ الخوارزمية الأصلية التي تفترض أن المرشحين يَرِدون وفق ترتيب عشوائي.

توطئة 3.5

كلفة التوظيف المتوقعة للإجراء RANDOMIZED-HIRE-ASSISTANT هي $O(c_h \ln n)$.

البرهان بعد إجراء التبديل في صفيفة الدخل، نكون قد وصلنا إلى حالة مماثلة لتلك التي درسناها في التحليل الاحتمالي لـ HIRE-ASSISTANT. ■

إن المقارنة بين التوطئة 2.5 والتوطئة 3.5 تُبْرز الفرق بين التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة. ففي التوطئة 2.5 نفترض فرضية محددة بخصوص الدخل، أما في التوطئة 3.5 فإننا لا نفترض مثل هذه

الفرضيات، إلاّ أن إدخال العشوائية على الدخل يستغرق زمنًا إضافيًا. حتى نبقى متوافقين مع مصطلحاتنا، تحدثنا في التوطئة 2.5 عن زمن التوظيف في الحالة الوسطى، وفي التوطئة 3.5 عن زمن التوظيف المتوقع. سنناقش فيما تبقّى من هذا المقطع بعض المسائل المتعلّقة بتبديل المدخلات عشوائيًا.

تبديل الصفيفات عشوائيًا

تُدْخِل العديدُ من الخوارزميات ذات العشوائيةِ المضافةِ، العشوائيةَ على الدخل بإجراء تبديل في صفيفة الدخل المعطاة. (هناك طرق أخرى لاستخدام السلوك العشوائي.) سنناقش هنا طريقتين للقيام بذلك. نفترض أن لدينا صفيفة A، وأنها تضمّ العناصر من 1 إلى n دون أن يؤثر ذلك على العمومية. هدفنا هنا هو توليد تبديل عشوائي لعناصر الصفيفة.

تتمثل إحدى الطرق الشائعة في إسناد أولوية عشوائية $P[i]$ لكل عنصر $A[i]$ من الصفيفة، ثم نفرز عناصر A وفق هذه الأولويات. فمثلاً، إذا كانت لدينا الصفيفة $A = \langle 1, 2, 3, 4 \rangle$، واخترنا الأولويات العشوائية $P = \langle 36, 3, 62, 19 \rangle$، فسنولّد الصفيفة $B = \langle 2, 4, 1, 3 \rangle$، إذ إن الأولوية الثانية هي الصغرى، تتبعها الرابعة، ثم الأولى، وأخيرًا الثالثة. نسمّي هذا الإجراء PERMUTE-BY-SORTING:

```
PERMUTE-BY-SORTING(A)
1  n = A.length
2  let P[1..n] be a new array
3  for i = 1 to n
4      P[i] = RANDOM(1, n^3)
5  sort A, using P as sort keys
```

يختار السطر 4 عددًا عشوائيًا بين 1 و n^3. نستخدم المجال من 1 إلى n^3 لنجعل وحدانية الأولويات في P أمرًا محتملاً. (يطلب إليك التمرين 5-3.5 أن تبرهن أن احتمال أن تكون كل الأولويات في P وحيدة هو على الأقل $1 - 1/n$، ويطلب إليك التمرين 6-3.5 أن تبيّن كيف تنجّز الخوارزميةَ وإن كانت أولويتان أو أكثر متماثلتين.) لنفترض أن كل الأولويات وحيدة.

الخطوة التي تستغرق وقتًا في هذا الإجراء هي الفرز في السطر 5. وسنرى في الفصل 8، أنه إذا استخدمنا فرزًا بالمقارنة، فإن الفرز يستغرق زمنًا $\Omega(n \lg n)$. يمكننا تحقيق هذا الحد الأدنى، فقد وجدنا أن الفرز بالدمج يستغرق زمنًا $\Theta(n \lg n)$. (ستتعرّف في الباب II طرائقَ أخرى للفرز بالمقارنة تستغرق $\Theta(n \lg n)$. يطلب إليك التمرين 4-3.8 أن تحل مسألة مشابهة جدًّا لفرز الأعداد في المجال بين 0 و $n^3 - 1$ في زمن $O(n)$.) إذا كانت $P[i]$، بعد الفرز، هي الأولوية الصغرى ذات الترتيب j، فسيكون $A[i]$ في الموقع j من الخرج. وبهذه الطريقة نحصل على تبديل. بقي أن نبرهن أن الإجراء يولّد ***تبديلاً عشوائيًا منتظمًا*** ***uniform random permutation***، أي يتساوى احتمال توليد أيٍّ من التباديل للأعداد من 1 إلى n.

توطئة 4.5

يولِّد الإجراء PRERMUTE-BY-SORTING تبديلاً عشوائيًّا منتظمًا للدخل، وذلك بافتراض أن كل الأولويات متمايزة فيما بينها.

البرهان نبدأ بدراسة التبديل الخاص الذي يتلقّى فيه كل عنصر $A[i]$ الأولوية الصغرى ذات الترتيب i. سنبيّن أن هذا التبديل يحدث باحتمال يساوي تمامًا $1/n!$. ليكن E_i، عندما $i = 1, 2, \ldots, n$، الحدثَ المقابلَ لأن يتلقّى العنصر $A[i]$ الأولوية الصغرى ذات الرقم i. ونريد أن نحسب احتمال وقوع الحدث مهما كانت قيمة i، وهو

$$\Pr\{E_1 \cap E_2 \cap E_3 \cap \cdots \cap E_{n-1} \cap E_n\} .$$

اعتمادًا على التمرين ت.2-5، يساوي هذا الاحتمال

$$\Pr\{E_1\} \cdot \Pr\{E_2 | E_1\} \cdot \Pr\{E_3 | E_2 \cap E_1\} \cdot \Pr\{E_4 | E_3 \cap E_2 \cap E_1\}$$
$$\cdots \Pr\{E_i | E_{i-1} \cap E_{i-2} \cap \cdots \cap E_1\} \cdots \Pr\{E_n | E_{n-1} \cap \cdots \cap E_1\} .$$

لدينا $\Pr\{E_1\} = 1/n$، لأنه احتمال أن تكون أولوية واحدة منتقاة عشوائيًّا من بين مجموعة من n أولوية هي الصغرى. ثمّ نلاحظ أن $\Pr\{E_2 | E_1\} = 1/(n-1)$ لأنه بعلمنا أن $A[1]$ أخذ الأولوية الصغرى، فإن لكل عنصر من العناصر $n-1$ المتبقية حظًّا متساويًا في أن يأخذ الأولويةَ الثانيةَ في الصغر. وعمومًا، عندما $i = 2, 3, \ldots n$، يكون لدينا $\Pr\{E_i | E_{i-1} \cap E_{i-2} \cap \cdots \cap E_1\} = 1/(n-i+1)$، وذلك لأنه إذا علمنا أن العناصر من $A[1]$ إلى $A[i-1]$ أخذت الأولويات الصغرى، التي عددها $i-1$ (وبالترتيب)، فإن لكل عنصر من العناصر $n-(i-1)$ المتبقية حظًّا متساويًا في أن تأخذ الأولوية الصغرى ذات الترتيب i. إذن لدينا

$$\Pr\{E_1 \cap E_2 \cap E_3 \cap \cdots \cap E_{n-1} \cap E_n\} = \left(\frac{1}{n}\right)\left(\frac{1}{n-1}\right)\cdots\left(\frac{1}{2}\right)\left(\frac{1}{1}\right)$$
$$= \frac{1}{n!} ,$$

وبذلك نكون قد بيّنا أن احتمال الحصول على التبديل المطابق هو $1/n!$.

يمكننا تعميم هذا البرهان على أي تبديل للأولويات. ليكن لدينا تبديل محدد $\sigma = \langle \sigma(1), \sigma(2), \ldots, \sigma(n) \rangle$ للمجموعة $\{1,2,\ldots,n\}$. ليكن r_i مرتبة الأولوية المسندة للعنصر $A[i]$، حيث يكون للعنصر ذي الأولوية الصغرى ذات الترتيب j، المرتبة j. إذا عرّفنا E_i ليكون الحدث المقابل لأن يتلقّى العنصر $A[i]$ الأولوية الصغرى ذات الترتيب $\sigma(i)$، أو $r_i = \sigma(i)$، فيمكن تطبيق البرهان السابق نفسه. إذن، إذا حسبنا احتمال الحصول على تبديل محدد، أيًّا كان، فإن الحساب مماثل للحساب السابق، ويكون احتمال الحصول على هذا التبديل هو أيضًا $1/n!$. ■

قد يعتقد المرء أنه يكفي لبرهان أن تبديلاً ما هو تبديل عشوائي منتظم، أن نبيّن أن احتمال أن ينتهي

أي عنصر $A[i]$ إلى الموقع j هو $1/n$. يبيّن التمرين 5.3-4 أن هذا الشرط الأضعف هو في الحقيقة غير كافٍ.

هناك طريقة أفضل لتوليد تبديل عشوائي وهو تبديل الصفيفة المعطاة في المكان، حيث يقوم الإجراء RANDOMIZE-IN-PLACE بذلك في زمن $O(n)$. يجري في التكرار i، اختيار العنصر $A[i]$ عشوائيًا من بين العناصر من $A[i]$ حتى $A[n]$، ولا يطرأ أي تغيير على $A[i]$ بعد هذا التكرار.

```
RANDOMIZE-IN-PLACE(A)
1  n = A.length
2  for i = 1 to n
3      swap A[i] with A[RANDOM(i, n)]
```

سنستخدم لامتغيّر حلقة لنبيّن أن الإجراء RANDOMIZE-IN-PLACE يولّد تبديلاً عشوائيًا منتظمًا. ليكن لدينا مجموعة من n عنصرًا. نسمي المتتالية التي تضم k عنصرًا من n عنصرًا دون تكرار: ***تبديل‒k*** (***k-permutation***). (انظر الملحق ت.) هناك $n!/(n-k)!$ تبديل $-\ k$ محتملاً.

توطئة 5.5

يَحسب الإجراء RANDOMIZE-IN-PLACE تبديلاً عشوائيًا منتظمًا.

البرهان نستخدم لامتغير الحلقة التالي:

قبل التكرار ذي الرقم i للحلقة **for** في السطرين 2-3، ومهما كان التبديل‒ $(i-1)$ للعناصر التي عددها n، تحتوي الصفيفة الجزئية $A[1..i-1]$ هذا التبديل‒ $(i-1)$ باحتمال يساوي $(n-i+1)!/n!$.

علينا أن نبيّن أن هذا اللامتغير صحيح قبل التكرار الأول للحلقة، وأن كل تكرار للحلقة يحافظ على هذا اللامتغير. ويجب أن نبين أيضًا أن هذا اللامتغير يقدم خاصية مفيدة تسمح بالتحقق من الصحة عندما تتوقف الحلقة.

الاستبداء: لندرس الحالة قبل التكرار الأول للحلقة، أي $i=1$. إن لامتغير الحلقة يعني أنه مهما كان التبديل‒0، فإن الصفيفة الجزئية $A[1..0]$ تحتوي هذا التبديل‒0 باحتمال يساوي $(n-i+1)!/n! = n!/n! = 1$. الصفيفة الجزئية $A[1..0]$ هي صفيفة جزئية فارغة، وأي تبديل‒0 لا يحوي أي عنصر، إذن تحتوي الصفيفة $A[1..0]$ أيَّ تبديل‒0، باحتمال 1، وبهذا يكون لامتغير الحلقة محققًا قبل التكرار الأول.

المحافظة على الشرط: نفترض أنه، قبل التكرار i مباشرةً، يظهر كل تبديل‒ $(i-1)$ في الصفيفة الجزئية $A[1..i-1]$ باحتمال يساوي $(n-i+1)!/n!$، وسوف نبيّن أنه بعد التكرار i، يظهر أيُّ تبديل‒ i

ممكنٍ في الصفيفة الجزئية $A[1..i]$ باحتمال $(n-i)!/n!$. إذن، بزيادة 1 على i للدخول في التكرار التالي سيبقى لامتغير الحلقة محققًا.

لندرس التكرار i. لنأخذ تبديل– i محددًا، ولنسمِّ عناصره $\langle x_1, x_2, \dots, x_i \rangle$. يتكّون هذا التبديل من تبديل– $(i-1)$ $\langle x_1, x_2, \dots, x_{i-1} \rangle$ متبوعٍ بالقيمة x_i التي تضعها الخوارزمية في $A[i]$. ليكن E_1 الحدث المتمثل في قيام التكرارات $i-1$ الأولى بإنشاء التبديل– $(i-1)$ المحدّد $\langle x_1, x_2, \dots, x_{i-1} \rangle$ في $A[1..i-1]$. اعتمادًا على لامتغيّر الحلقة، يكون $\Pr\{E_1\} = (n-i+1)!/n!$. ليكن E_2 الحدث المتمثل في أن يضع التكرار i العنصر x_i في الموقع $A[i]$. إن التبديل– i $\langle x_1, \dots, x_i \rangle$ في الصفيفة $A[1..i]$ يَظهر تمامًا مع حدوث كلٍّ من E_1 و E_2، ولهذا السبب نريد حساب $\Pr\{E_2 \cap E_1\}$. باستخدام المعادلة (ت.14)، يكون لدينا

$$\Pr\{E_2 \cap E_1\} = \Pr\{E_2 | E_1\} \Pr\{E_1\} \ .$$

إن الاحتمال $\Pr\{E_2 | E_1\}$ يساوي $1/(n-i+1)$ لأن السطر 3 من الخوارزمية يختار x_i عشوائيًا من بين $n-i+1$ قيمة في المواقع $A[i..n]$. إذن لدينا

$$\begin{aligned}
\Pr\{E_2 \cap E_1\} &= \Pr\{E_2 | E_1\} \Pr\{E_1\} \\
&= \frac{1}{n-i+1} \cdot \frac{(n-i+1)!}{n!} \\
&= \frac{(n-i)!}{n!}
\end{aligned}$$

الإنهاء: في النهاية، $i = n+1$، ويكون لدينا أن الصفيفة الجزئية $A[1..n]$ هي تبديل– n معطى باحتمال $(n-(n+1)+1)!/n! = 0!/n! = 1/n!$.

وبهذا، تنشئ RANDOMIZE-IN-PLACE تبديلاً عشوائيًا منتظمًا. ■

إن الخوارزمية ذات العشوائية المضافة هي في كثير من الأحيان أبسط الطرق وأكثرها فعالية لحل مسألة ما. سنستخدم الخوارزميات ذات العشوائية المضافة في مواقع عديدة في هذا الكتاب.

تمارين

1-3.5

يحتج الأستاذ مارسو Marceau على لامتغيّر الحلقة المستخدم في برهان التوطئة 5.5. إنه يتساءل فيما إذا كان محققًا قبل التكرار الأول. محاكمته مبنية على أنه بإمكان المرء أن يصرّح ببساطة أن صفيفة جزئية فارغة لا تحتوي أي تبديل–0. ولهذا السبب، فإن احتمالَ أن تحتوي صفيفة جزئية فارغة على تبديل–0 معدومٌ، وهذا ما يجعل لامتغيّر الحلقة غير محقّق قبل التكرار الأوّل. أعِدْ كتابة الإجراء RANDOMIZE-IN-PLACE بحيث

يكون لامتغيّر الحلقة الموافق له محققًا على صفيفة جزئية غير فارغة قبل الدخول في التكرار الأوّل، وعدّل برهان التوطئة 5.5 لإجرائيتك.

2-3.5

قرّر الأستاذ كيلب Kelp أن يكتب إجراءً ينشئ عشوائيًا أي تبديل باستثناء التبديل المطابق. وهو يقترح الإجراء التالي:

```
PERMUTE-WITHOUT-IDENTITY(A)
1  n = A.length
2  for i = 1 to n - 1
3        swap A[i] with A[RANDOM(i + 1, n)]
```

هل يقوم هذا الرماز بما يبغيه الأستاذ كيلب؟

3-3.5

لنفترض أنه، بدلاً من مبادلة العنصر $A[i]$ بعنصر عشوائي من الصفيفة الجزئية $A[i..n]$، فإننا نبادله بعنصر عشوائي من أي موقع في الصفيفة:

```
PERMUTE-WITH-ALL(A)
1   n = A.length
2   for i = 1 to n
3        swap A[i] with A[RANDOM(1, n)]
```

هل يُنتج هذا الرماز تبديلاً عشوائيًا منتظمًا؟ لماذا أو لمَ لا؟

4-3.5

يقترح الأستاذ أرمسترونغ Armstrong الإجراء التالي لتوليد تبديل عشوائي منتظم:

```
PERMUTE-BY-CYCLIC(A)
1  n = A.length
2  let B[1..n] be a new array
3  offset = RANDOM(1, n)
4  for i = 1 to n
5       dest = i + offset
6       if dest > n
7            dest = dest - n
8       B[dest] = A[i]
9  return B
```

بيّن أن كل عنصر $A[i]$ له احتمال $1/n$ لينتهي في أي موقع محدّد في B. ثمّ بيّن أن الأستاذ أرمسترونغ مخطئ، وذلك بأن تبيّن أن التبديل الناتج ليس تبديلاً عشوائيًا منتظمًا.

★ *5-3.5*

برهن أن احتمال أن تكون كل العناصر وحيدة في الصفيفة P في الإجراء PERMUTE-BY-SORTING، هو على الأقل $1 - 1/n$.

6-3.5

اشرح كيف يمكن تنجيز الخوارزمية PERMUTE-BY-SORTING لتعالج حالة تَساوي أولويتين أو أكثر. أي يجب على خوارزميتك أن تُنتج تبديلاً عشوائيًا منتظمًا ولو كانت هناك أولويتان أو أكثر متساويتين.

7-3.5

افترض أننا نريد إنشاء ***عينة عشوائية random sample*** من المجموعة $\{1, 2, 3, \ldots, n\}$، أي مجموعة جزئية من m عنصرًا، حيث $0 \le m \le n$، بحيث يكون احتمال إنشاء أية مجموعة من المجموعات الجزئية ذات m عنصرًا متساويًا. تتمثل إحدى الطرق بجعل $A[i] = i$، حيث $i = 1, 2, 3, \ldots, n$، ثم استدعاء RANDOMIZE-IN-PLACE(A)، وأخذ العناصر m الأولى من الصفيفة. قد تقوم هذه الطريقة باستدعاء الإجراء RANDOM n مرة. إذا كانت n أكبر كثيرًا من m، يمكننا إنشاء عينة عشوائية بعدد أقل من الاستدعاءات لـ RANDOM. بيّن أن الإجراء العودي التالي يعيد المجموعة الجزئية S المكونة من m عنصرًا من $\{1, 2, 3, \ldots, n\}$، حيث تكون كل المجموعات الجزئية الممكنة متساوية الاحتمال، وذلك بالاعتماد على m استدعاء فقط لـ RANDOM.

```
RANDOM-SAMPLE(m, n)
1  if i == 0
2      return Ø
3  else S = RANDOM-SAMPLE(m - 1, n - 1)
4      i = RANDOM(1, n)
5      if i ∈ S
6          S = S ∪ {n}
7      else S = S ∪ {i}
8      return S
```

★ 4.5 التحليل الاحتمالي واستخدامات إضافية للمؤشرات العشوائية

يتعمق هذا المقطع المتقدّم في شرح التحليل الاحتمالي عن طريق أربعة أمثلة. يحدّد الأول احتمال أن يشترك شخصان من بين k شخصًا مجتمعين في غرفةٍ بيوم ميلادهما. يدرس المثال الثاني ما يحدث عندما نرمي كرات عشوائيًا في سلاّت. ويستكشف الثالث ضربات الحظ "Streaks" المتمثلة في الحصول على وجوه متتالية عند رمي قطعة نقد. ويحلّل المثال الأخير نموذجًا معدّلاً لمسألة التوظيف عليك أن تتخذ فيه القرارات دون أن تقابل فعليًا كل المرشحين.

1.4.5 متناقضة يوم الميلاد

مثالنا الأول هو ***متناقضة يوم الميلاد birthday paradox***. ما عدد الأشخاص الذين يجب أن يُوجَدوا في مكان واحد حتى يكون احتمال أن يشترك اثنان منهما في يوم ميلادهما يساوي 50%؟ والجواب على عكس ما قد تتوقعه، هو عدد قليل. وهنا يكمن التناقض فالعدد في الواقع، وكما سنرى الآن أقل بكثير من عدد أيام السنة أو حتى من نصف عدد أيام السنة.

سنقوم، للإجابة على هذا السؤال، بالإشارة إلى الأشخاص في الغرفة بأعداد طبيعيّة $1, 2, .., k$، حيث k هو عدد الأشخاص الكلّي في الغرفة. سنتجاهل مسألة السنة الكبيسة ونفترض أن عدد الأيام في كل السنوات متساوٍ ويساوي $n = 365$. ليكن b_i، حيث $i = 1,2,..,k$، رقم يوم ميلاد الشخص i من أيام السنة، حيث $1 \le b_i \le n$. نفترض أيضًا أن أيام الميلاد موزعة توزيعًا منتظمًا على كل أيام السنة n، بحيث يكون $\Pr\{b_i = r\} = 1/n$ لكل $i = 1, 2, ..., k$ و $r = 1, 2, ..., n$.

إن احتمال أن يكون لشخصين، i و j مثلاً، يومُ ميلادٍ مشتركٌ يتعلّق باستقلال الانتقاء العشوائي لأيام الميلاد. ونفترض من الآن فصاعدًا أن أيام الميلاد مستقلة فيما بينها، وهكذا يكون احتمال أن يقع يوم ميلاد i ويوم ميلاد j معًا في اليوم r هو

$$\begin{aligned}\Pr\{b_i = r \text{ and } b_j = r\} &= \Pr\{b_i = r\} \Pr\{b_j = r\} \\ &= 1/n^2 .\end{aligned}$$

ومنه، يكون احتمال أن يقعا معًا في اليوم نفسه هو

$$\begin{aligned}\Pr\{b_i = b_j\} &= \sum_{r=1}^{n} \Pr\{b_i = r \text{ and } b_j = r\} \\ &= \sum_{r=1}^{n} (1/n^2) \\ &= 1/n . \qquad (6.5)\end{aligned}$$

يمكننا أن نرى ببساطة أكثر، أنه ما إن يتم اختيار b_i، فإن احتمال أن يتم اختيار b_j ليكون اليوم b_i نفسه هو $1/n$. إذن، احتمال أن يتماثل يوما ميلاد i و j هو نفسه احتمال أن يقع يوم ميلاد أحدهما في يوم محدَّد. ولكن لاحظ أنه هذه المصادفة مرتبطة في الواقع بافتراضنا أن أيام الميلاد مستقلّة فيما بينها.

يمكننا أن ندرس احتمال أن يكون، على الأقل، لاثنين من k شخصًا، يوم الميلاد نفسه بالنظر إلى الحدث المتمم. إن احتمال أن يتماثل على الأقل اثنين من أيام الميلاد هو 1 مطروحًا منه احتمال أن تكون جميع أيام الميلاد مختلفة. إن الحدث المتمثل في أن يكون لـ k شخصًا أيام ميلاد متمايزة هو

$$B_k = \bigcap_{i=1}^{k} A_i \ ,$$

حيث A_i هو الحدث المتمثل في أن يكون يوم ميلاد i مختلفًا عن يوم ميلاد الشخص j لجميع قيم $j < i$. ولما كان بمقدورنا أن نكتب $B_k = A_k \cap B_{k-1}$، فإننا نحصل من المعادلة (ت.16) على العلاقة العودية

$$\Pr\{B_k\} = \Pr\{B_{k-1}\} \Pr\{A_k | B_{k-1}\} \ , \tag{7.5}$$

حيث نأخذ $\Pr\{B_1\} = \Pr\{A_1\} = 1$ باعتباره شرطًا ابتدائيًّا. وبعبارة أخرى، إن احتمال أن يكون $b_1, b_2, \ldots, b_k$ أيام ميلاد متمايزة هو احتمال أن تكون $b_1, b_2, \ldots, b_{k-1}$ أيامًا متمايزة مضروبًا باحتمال أن يكون $b_k \neq b_i$ حيث $i = 1, 2, \ldots, k-1$ علمًا أن $b_1, b_2, \ldots, b_{k-1}$ متمايزة.

إذا كانت الأيام $b_1, b_2, \ldots, b_{k-1}$ متمايزة، فإن الاحتمال الشرطي ليكون $b_k \neq b_i$ عندما $i = 1, 2, \ldots, k-1$ هو $\Pr\{A_k | B_{k-1}\} = (n-k+1)/n$، إذ إن $n-(k-1)$ لم تُؤخذ من n يومًا. نطبق العلاقة العودية (7.5) تكراريًّا فنحصل على

$$\begin{aligned}
\Pr\{B_k\} &= \Pr\{B_{k-1}\} \Pr\{A_k | B_{k-1}\} \\
&= \Pr\{B_{k-2}\} \Pr\{A_{k-1} | B_{k-2}\} \Pr\{A_k | B_{k-1}\} \\
&\ \vdots \\
&= \Pr\{B_1\} \Pr\{A_2 | B_1\} Pr\{A_3 | B_2\} \cdots \Pr\{A_k | B_{k-1}\} \\
&= 1 \cdot \left(\frac{n-1}{n}\right)\left(\frac{n-2}{n}\right) \cdots \left(\frac{n-k+1}{n}\right) \\
&= 1 \cdot \left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right) .
\end{aligned}$$

تعطينا المتراجحة (12.3)، $1 + x \leq e^x$:

$$\begin{aligned}
\Pr\{B_k\} &\leq e^{-1/n} e^{-2/n} \ \ldots \ e^{-(k-1)/n} \\
&= e^{-\sum_{i=1}^{k-1} i/n} \\
&= e^{-k(k-1)/2n} \\
&\leq 1/2
\end{aligned}$$

عندما $-k(k-1)/2n \leq \ln(1/2)$. إن احتمال أن تكون كل أيام الميلاد، وعددها k، متمايزة هو على الأكثر $1/2$ عندما $k(k-1) \geq 2n \ln 2$، أو بحل المعادلة التربيعيّة عندما $k \geq (1 + \sqrt{1 + (8 \ln 2) n})/2$. ففي حالة $n = 365$، يجب أن يكون لدينا $k \geq 23$. إذن إذا كان هناك على الأقل 23 شخصًا في غرفة، فإن احتمال أن يشترك على الأقل شخصان بيوم ميلادهما هو 1/2 على الأقل. أما على كوكب المريخ، فإن السنة تبلغ 669 يومًا مريخيًّا؛ لذلك يجب أن يكون هناك 31 مريخيًّا لنحصل على الأثر نفسه.

تحليل باستخدام المؤشرات العشوائية

يمكننا استخدام المؤشرات العشوائية لنقدّم تحليلاً أبسط، ولكنه تقريبيّ، لمتناقضة يوم الميلاد. نعرّف، لكل ثنائية (i,j) من k شخصًا في الغرفة، المؤشر العشوائي X_{ij} ، حيث $1 \le i < j \le k$، بـ

$$X_{ij} = \text{I}\{\text{person } i \text{ and person } j \text{ have the same birthday}\}$$
$$= \begin{cases} 0 & \text{if person } i \text{ and person } j \text{ have the same birthday}, \\ 1 & \text{otherwise}. \end{cases}$$

[أي X_{ij} يساوي 1 إذا كان الشخص i والشخص j يشتركان في يوم ميلادهما.]

اعتمادًا على المعادلة (6.5)، نعلم أن احتمال أن يكون لشخصين يوم الميلاد نفسه هو $1/n$، إذن بالاعتماد على التوطئة 1.5 لدينا

$$\text{E}[X_{ij}] = \Pr\{\text{person } i \text{ and person } j \text{ have the same birthday}\}$$
$$= 1/n .$$

إذا أخذنا X ليكون المتحول العشوائي الذي يَعُدُّ أزواج الأشخاص الذين يتشارك كل زوج منهم بيوم ميلاد واحد، يكون لدينا

$$X = \sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{ij} .$$

وبحساب التوقع للطرفين، وبتطبيق خطيّة التوقع نحصل على

$$\begin{aligned} \text{E}[X] &= \text{E}\left[\sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{ij}\right] \\ &= \sum_{i=1}^{k} \sum_{j=i+1}^{k} \text{E}[X_{ij}] \\ &= \binom{k}{2}\frac{1}{2} \\ &= \frac{k(k-1)}{2n} . \end{aligned}$$

إذن، عندما $k(k-1) \ge 2n$، يكون العدد المتوقع لأزواج الأشخاص الذين لهم يوم ميلاد مشترك مساويًا 1 على الأقل. أي إذا كان لدينا في الغرفة على الأقل $\sqrt{2} + 1$ شخصًا، نتوقع أن يكون هناك على الأقل شخصان يشتركان في يوم ميلادهما. فإذا كان $n = 365$، و $k = 28$، فإن العدد المتوقع لأزواج الأشخاص الذين لهم يوم ميلاد مشترك هو $(28 \cdot 27)/(2 \cdot 365) \approx 1.0356$. إذن، بوجود 28 شخصًا على الأقل، نتوقع أن نجد على الأقل زوجًا من الأشخاص الذين يتشاركون في يوم ميلادهم. أما على المريخ، حيث يبلغ

طول السنة 669 يومًا مريخيًا، فنحتاج على الأقل إلى 38 مريخيًا.

لقد حددنا باستخدام طريقة التحليل الأولى، المعتمدة على الاحتمالات فقط، عدد الأشخاص اللازم حتى يتجاوز احتمال وجود زوج يتشارك في يوم الميلاد القيمة 1/2، وباستخدام طريقة التحليل الثانية، التي استخدمت المؤشرات العشوائية، حددنا العدد اللازم حتى يكون العدد المتوقع لأيام الميلاد المشتركة يساوي 1. وعلى الرغم من أن عدد الأشخاص الدقيق يختلف في الحالتين، إلا أنهما متماثلان بالمقاربة: $\Theta(\sqrt{n})$.

2.4.5 الكرات والسلال

سنناقش عملية الرمي العشوائي لكراتٍ متماثلةٍ في b سلّة مرقّمة $1,2,..,b$، حيث تكون الرميات مستقلّةً فيما بينها، ويكون احتمال أن تنتهي الكرة في أية سلة - عند كل رمية – متساويًا. إن احتمال أن تستقر الكرة المرميّة في أية سلة محدّدة هو $1/b$. أي إنَّ عمليّة رمي الكرات هي متتالية من تجارب برنولية Bernoulli trails (انظر الملحق ت.4) باحتمال نجاح يساوي $1/b$، حيث يعني النجاح هنا أن تقع الكرة في السلّة المحدّدة. إن هذا النموذج ذو فائدة خاصة عند تحليل التلبيد hashing (انظر الفصل 11)، ويمكننا أن نجيب عن العديد من الأسئلة المثيرة للاهتمام بخصوص عملية رمي الكرات. (تطرح المسألة ت-1 أسئلة إضافية بشأن الكرات والسلال.)

ما عدد الكرات التي تقع في سلةٍ محددة؟ إن عدد الكرات الذي يقع في سلة محددة يتبع التوزيع الثنائي الحد $b(k;n,1/b)$. إذا رُميت n كرة، فإن المعادلة (ت.37) تقرّر أن العدد المتوقع للكرات التي تقع في سلة محددة هو n/b.

ما عدد الكرات التي يجب أن نرميها وسطيًا حتى تحتوي سلة محددة على كرةٍ واحدة؟ إن عدد الرميات اللازم حتى تتلقى سلةٌ محددةٌ كرةً ما يتبع التوزيع الهندسي باحتمال $1/b$، واعتمادًا على المعادلة (ت.32)، يكون عدد الرميات المتوقع حتى حصول النجاح هو $1/(1/b) = b$.

ما عدد الكرات الذي يجب أن نرميها حتى تحتوي كل سلّة على كرة واحدة على الأقل؟ نسمّي الرمية التي توقع كرةً في سلةٍ فارغةٍ "هدفًا hit". ونريد أن نعرف عدد الرميات المتوقع n اللازم للحصول على b هدفًا.

يمكن استخدام الأهداف لتجزيء n رمية إلى مراحل. تتكون المرحلة i من الرميات بعد الهدف $i-1$ وحتى الحصول على الهدف i. وتتكوّن المرحلة الأولى من رمية واحدة، إذ إنه من المؤكد أننا سنحقّق هدفًا عندما تكون كل السلاّت فارغة. عند كل رمية في المرحلة i، يكون لدينا $i-1$ سلّة فيها كرات و $b-i+1$ سلّة فارغة. إذن، أيًّا كانت الرمية في المرحلة i، يكون احتمال الحصول على هدف مساويًا $(b-i+1)/b$.

نسمّي n_i عدد الرميات في المرحلة i، فيكون عدد الرميات اللازم للحصول على b هدفًا هو $n = \sum_{i=1}^{b} n_i$. إن كل متحوّل عشوائي n_i يتبع توزيعًا هندسيًّا باحتمال نجاحٍ يساوي $(b-i+1)/b$.

واعتمادًا على المعادلة (ت.32) يكون

$$\mathrm{E}[n_i] = \frac{b}{b-i+1} \ .$$

واعتمادًا على خطيّة التوقع، لدينا

$$\begin{aligned}
\mathrm{E}[n] &= \mathrm{E}\left[\sum_{i=1}^{b} n_i\right] \\
&= \sum_{i=1}^{b} \mathrm{E}[n_i] \\
&= \sum_{i=1}^{b} \frac{b}{b-i+1} \\
&= b\sum_{i=1}^{b} \frac{1}{i} \\
&= b(\ln b + O(1)) \ . && \text{(اعتمادًا على المعادلة (أ.7))}
\end{aligned}$$

نستنتج مما سبق أننا بحاجة إلى $b \ln b$ رمية تقريبًا قبل أن نتوقع أن يكون هناك كرة في كل سلة. تُعرَف هذه المسألة أيضًا باسم ***مسألة جامع القسائم*** ***coupon collector's problem***، التي تنص على أن الشخص الذي يحاول جمع قسيمةٍ من كل نوع من b قسيمة مختلفة، عليه أن يُحْرِزَ $b \ln b$ قسيمة تقريبًا يُحصِّلها عشوائيًّا لكي ينجح في مسعاه.

3.4.5 ضربات الحظ

لنفترض أنك ترمي قطعة نقد عادلة n مرّة. ما هي أطول ضربة حظ streak متمثلة في سلسلة وجوه متتالية تتوقع الحصول عليها؟ الجواب هو $\Theta(\lg n)$، كما يبيّن التحليل التالي.

نبرهن أولاً أن الطول المتوقع لأطول ضربة حظ هو $O(\lg n)$. إن احتمال أن يكون ناتج رمي كل قطعة وجهًا هو 1/2. ليكن A_{ik} الحدث المتمثل في أن ضربة حظ طولها على الأقل k تبدأ بالرمية رقم i، أو بدقة أكبر هو الحدث المتمثل في أن k رمية متتالية $i, i+1, \ldots, i+k-1$ تعطي وجوهًا فقط، حيث $1 \le k \le n$ و $1 \le k \le n-k+1$. ولمّا كانت رميات قطعة النقد كلها مستقلة فيما بينها، أيًّا كان الحدث المعطى A_{ik}، فإن احتمال أن تكون كل الرميات، وعددها k، وجوهًا هو

$$\Pr\{A_{ik}\} = 1/2^k \ . \tag{8.5}$$

فإذا كان $k = 2\lceil \lg n \rceil$، فإن

$$
\begin{aligned}
\Pr\{A_{i,2\lceil \lg n\rceil}\} &= 1/2^{2\lceil \lg n\rceil} \\
&\le 1/2^{2\lg n} \\
&= 1/n^2 ,
\end{aligned}
$$

وبذلك يكون احتمالُ حدوث ضربة حظ طولها على الأقل $2\lceil \lg n\rceil$ بدءًا من الموقع i صغيرًا جدًا. ولما كان عدد المواقع عندما تبدأ مثل هذه الضربة هو $n - 2\lceil \lg n\rceil + 1$ موقعًا على الأكثر، فإن احتمال الحصول على ضربة حظ طولها على الأقل $2\lceil \lg n\rceil$ وجهًا تبدأ في أي موقع ممكن هو

$$
\begin{aligned}
\Pr\left\{\bigcup_{i=1}^{n-2\lceil \lg n\rceil+1} A_{i,2\lceil \lg n\rceil}\right\} &\le \sum_{i=1}^{n-2\lceil \lg n\rceil+1} 1/n^2 \\
&< \sum_{i=1}^{n} 1/n^2 \\
&= 1/n , \qquad (9.5)
\end{aligned}
$$

وذلك اعتمادًا على متراجحة بول (ت.19)، حيث إن احتمال اجتماع عدد من الأحداث يساوي على الأكثر مجموع احتمالات الأحداث الفردية. (لاحظ أن متراجحة بول محققة وإن لم تكن الأحداث مستقلةً.)

نستخدم الآن المتراجحة (9.5) لنحدّ طول أطول ضربة حظ. ليكن L_j الحدث المتمثل في أن أطول ضربة حظ طولها j تمامًا، حيث $j = 0, 1, 2, \ldots, n$، وليكن L طول أطول ضربة حظ. لدينا اعتمادًا على تعريف القيمة المتوقعة،

$$
\mathrm{E}[L] = \sum_{j=0}^{n} j \Pr\{L_j\} . \qquad (10.5)
$$

يمكننا محاولة حساب هذا المجموع باستخدام حدودٍ عليا على كل $\Pr\{L_j\}$ كتلك المحسوبة في المتراجحة (9.5)، ولكن لسوء الحظ، قد تعطي هذه الطريقة حدودًا ضعيفة. إلا أنه بإمكاننا الاعتماد على بعض الحدس المكتسب من التحليل السابق لنحصل على حدٍّ جيد. نلاحظ، بدايةً، أنه لا توجد حدود إفرادية في المجموع الوارد في المعادلة (10.5) يكون فيها كلٌّ من العاملين j و $\Pr\{L_j\}$ كبيرًا. لماذا؟ لأنه عندما يكون $j \ge 2\lceil \lg n\rceil$، فإن $\Pr\{L_j\}$ يكون صغيرًا جدًا، وعندما يكون $j < 2\lceil \lg n\rceil$، فإن قيمة j تكون صغيرة على نحو لا بأس به. وبعد دراسة أدق، نلاحظ أن الأحداث L_j منفصلة عندما $j = 0, 1, \ldots, n$، ومن ثَمَّ فإن احتمال أن تبدأ ضربة حظ طولها على الأقل $2\lceil \lg n\rceil$ وجهًا في أي موقع هو: $\sum_{j=2\lceil \lg n\rceil}^{n} \Pr\{L_j\}$. ومن المعادلة (9.5)، يكون لدينا $\sum_{j=2\lceil \lg n\rceil}^{n} \Pr\{L_j\} < 1/n$.

وإذا لاحظنا أيضًا أن $\sum_{j=0}^{n} \Pr\{L_j\} = 1$، يكون لدينا $\sum_{j=0}^{2\lceil \lg n\rceil-1} \Pr\{L_j\} \le 1$. إذن نحصل على

$$\begin{aligned}
\mathrm{E}[L] &= \sum_{j=0}^{n} j \Pr\{L_j\} \\
&= \sum_{j=0}^{2\lceil\lg n\rceil-1} j \Pr\{L_j\} + \sum_{j=2\lceil\lg n\rceil}^{n} j \Pr\{L_j\} \\
&< \sum_{j=0}^{2\lceil\lg n\rceil-1} (2\lceil\lg n\rceil) \Pr\{L_j\} + \sum_{j=2\lceil\lg n\rceil}^{n} n \Pr\{L_j\} \\
&= 2\lceil\lg n\rceil \sum_{j=0}^{2\lceil\lg n\rceil-1} \Pr\{L_j\} + n \sum_{j=2\lceil\lg n\rceil}^{n} \Pr\{L_j\} \\
&< 2\lceil\lg n\rceil \cdot 1 + n \cdot (1/n) \\
&< O(\lg n)
\end{aligned}$$

إن احتمال أن تتجاوز ضربة الحظ $r\lceil\lg n\rceil$ وجهًا تتناقص بسرعة مع تزايد r. فعندما يكون $r \geq 1$، فإن احتمال أن تبدأ ضربة حظ طولها $r\lceil\lg n\rceil$ وجهًا بدءًا من الموقع i هو

$$\begin{aligned}
\Pr\{A_{i,r\lceil\lg n\rceil}\} &= 1/2^{r\lceil\lg n\rceil} \\
&\leq 1/2^r .
\end{aligned}$$

وهكذا، فإن احتمال أن يبلغ طول أطول ضربة حظ $r\lceil\lg n\rceil$ على الأقل هو $n/n^r = 1/n^{r-1}$ على الأكثر، أو بعبارةٍ مكافئة: احتمال أن يكون طول أطول ضربة حظ أقل من $r\lceil\lg n\rceil$ هو $1 - 1/n^{r-1}$. لنأخذ $n = 1000$ رمية، مثالاً على ذلك، إن احتمال الحصول على ضربة حظ طولها على الأقل $2\lceil\lg n\rceil = 20$ وجهًا هو على الأكثر $1/n = 1/1000$. وحظوظ الحصول على ضربة حظ تتجاوز $3\lceil\lg n\rceil = 30$ وجهًا هي على الأكثر $1/n^2 = 1/1{,}000{,}000$.

سنبرهن الآن حدًا أدنى متممًا: إن الطول المتوقع لأطول ضربة حظ في n رمية هو $\Omega(\lg n)$. ولبرهان هذا الحد، نبحث عن ضربات حظ طولها s بتجزيء الرميات، التي عددها n، إلى n/s مجموعة تقريبًا من s رمية. فإذا اخترنا $s = \lfloor(\lg n)/2\rfloor$، أمكننا أن نبيّن أنه من المحتمل أن تأتي كل الرميات في إحدى هذه المجموعات وجوهًا، ونستنتج من هذا أنه من المحتمل أن تكون طول أطول ضربة حظ هو على الأقل $s = \Omega(\lg n)$. ونبيّن فيما بعد أن الطول المتوقع لأطول ضربة حظ هو $\Omega(\lg n)$.

نجزئ الـ n رميةً لقطعة النقود إلى $\lfloor n/\lfloor(\lg n)/2\rfloor\rfloor$ مجموعة على الأقل من $\lfloor(\lg n)/2\rfloor$ رمية متتالية، ونعطي حدًّا لاحتمال ألاّ يكون هناك أية مجموعة كلها وجوه. واعتمادًا على المعادلة (8.5)، يكون احتمال أن تأتي كل رميات المجموعة التي تبدأ في الموقع i وجوهًا هو

$$\Pr\{A_{i,\lfloor(\lg n)/2\rfloor}\} = 1/2^{\lfloor(\lg n)/2\rfloor}$$
$$\geq 1/\sqrt{n} \; .$$

إذن، احتمال ألا تبدأ ضربة حظ طولها على الأقل $\lfloor(\lg n)/2\rfloor$ وجهًا من الموقع i هو على الأكثر $1 - 1/\sqrt{n}$. ولمّا كانت المجموعات $\lfloor n/\lfloor(\lg n)/2\rfloor\rfloor$ مكوَّنةً من رميات متمايزة ومستقلّة فيما بينها، فإن احتمال أن تفشل كل مجموعة من هذه المجموعات في أن تكون ضربة حظ طولها $\lfloor(\lg n)/2\rfloor$ هو على الأكثر

$$\begin{aligned}
\left(1 - 1/\sqrt{n}\right)^{\lfloor n/\lfloor(\lg n)/2\rfloor\rfloor} &\leq \left(1 - 1/\sqrt{n}\right)^{n/\lfloor(\lg n)/2\rfloor - 1} \\
&\leq \left(1 - 1/\sqrt{n}\right)^{2n/\lg n - 1} \\
&\leq e^{-(2n/\lg n - 1)/\sqrt{n}} \\
&= O\left(e^{-\lg n}\right) \\
&= O(1/n) \; .
\end{aligned}$$

وقد استخدمنا في هذا التعليل المتراجحة (12.3)، $1 + x \leq e^x$، ومتراجحةً قد ترغب في التحقق منها، وهي أن $(2n/\lg n - 1)/\sqrt{n} \geq \lg n$ عندما n كبيرة كفاية.

إذن، احتمال أن تساوي أطول ضربة حظ الطول $\lfloor(\lg n)/2\rfloor$ أو تتجاوزه هو

$$\sum_{j=\lfloor(\lg n)/2\rfloor+1}^{n} \Pr\{L_j\} \geq 1 - O(1/n) \; . \tag{11.5}$$

يمكننا الآن أن نحسب حدًّا أدنى على الطول المتوقع لأطول ضربة حظ، بأن نبدأ بالمعادلة (10.5)، ونقوم بالعمليات بطريقة مشابهة لما قمنا به لحساب الحد الأعلى:

$$\begin{aligned}
\mathrm{E}[L] &= \sum_{j=0}^{n} \Pr\{L_i\} \\
&= \sum_{j=0}^{\lfloor(\lg n)/2\rfloor - 1} j\Pr\{L_j\} + \sum_{j=\lfloor(\lg n)/2\rfloor}^{n} j\Pr\{L_j\} \\
&\geq \sum_{j=0}^{\lfloor(\lg n)/2\rfloor - 1} 0 \cdot \Pr\{L_j\} + \sum_{j=\lfloor(\lg n)/2\rfloor}^{n} \lfloor(\lg n)/2\rfloor \Pr\{L_j\} \\
&= 0 \cdot \sum_{j=0}^{\lfloor(\lg n)/2\rfloor - 1} \Pr\{L_j\} + \lfloor(\lg n)/2\rfloor \sum_{j=\lfloor(\lg n)/2\rfloor}^{n} \Pr\{L_j\}
\end{aligned}$$

$$\geq 0 + \lfloor (\lg n)/2 \rfloor (1 - O(1/n)) \qquad \text{(من المتراجحة (11.5))}$$
$$= \Omega(\lg n) .$$

بإمكاننا، كما كان الحال في متناقضة يوم الميلاد، الحصول على تحليل أبسط ولكنه تقريبي باستخدام المؤشرات العشوائية. ليكن $X_{ik} = \mathrm{I}\{A_{ik}\}$ المؤشر العشوائي المقابل لحدوث ضربة حظ طولها على الأقل k بدءًا من الرمية رقم i. لحساب العدد الكلي لمثل هذه الضربات، نعرّف

$$X = \sum_{i=1}^{n-k+1} X_{ik} .$$

وبأخذ توقعي الطرفين، وباستخدام خطيّة التوقع، يكون لدينا

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-k+1} X_{ik}\right] \\
&= \sum_{i=1}^{n-k+1} \mathrm{E}[X_{ik}] \\
&= \sum_{i=1}^{n-k+1} \Pr\{A_{ik}\} \\
&= \sum_{i=1}^{n-k+1} 1/2^k \\
&= \frac{n-k+1}{2^k} .
\end{aligned}$$

يمكننا، بتعويض قيم مختلفة لـ k، حساب العدد المتوقع لضربات الحظ ذات الطول k. إذا كان هذا العدد كبيرًا (أكبر بكثير من 1)، فهذا يعني أن من المتوقع حدوث عدّة ضربات حظ طولها k، واحتمال حدوث واحدة مرتفع. وإذا كان هذا العدد صغيرًا (أقل بكثير من 1)، فهذا يعني أن من المتوقع حدوث عددٍ قليلٍ جدًّا من ضربات الحظ ذات الطول k، واحتمال حدوث ضربة حظ واحدة منخفض. إذا كان $k = c \lg n$، حيث c ثابت موجب ما، نحصل على

$$\begin{aligned}
\mathrm{E}[X] &= \frac{n - c\lg n + 1}{2^{c \lg n}} \\
&= \frac{n - c\lg n + 1}{n^c} \\
&= \frac{1}{n^{c-1}} - \frac{(c\lg n - 1)/n}{n^{c-1}} \\
&= \Theta(1/n^{c-1}) .
\end{aligned}$$

إذا كان c كبيرًا، فإن العدد المتوقع لضربات الحظ ذات الطول $c \lg n$ صغير جدًّا، ونستنتج أنه من غير المحتمل أن تحدث. من جهة أخرى، إذا كان $c = 1/2$، فنحصل على $\mathrm{E}[X] = \Theta(1/n^{1/2-1}) = \Theta(1/n^{1/2})$ ، ونتوقع أنه سيكون هناك عدد كبير من ضربات الحظ ذات الطول $(1/2)\lg n$. لذا، فمن المتوقع جدًّا حدوث ضربة حظ واحدة من هذا الطول. بمقدورنا الآن، من هذه التوقعات الخشنة فقط، أن نستنتج أن توقع طول أطول ضربة حظ هو $\Theta(\lg n)$.

4.4.5 مسألة التوظيف على الخط

سندرس، في هذا المثال الأخير، شكلاً معدّلاً من مسألة التوظيف. لنفترض الآن أننا لا نريد أن نقابل كل المرشحين للعثور على أفضل مرشح بينهم. ولا نريد أيضًا أن نوظف ونسرّح عندما نقع على متقدمين أفضل فأفضل. عوضًا عن ذلك، نريد أن نكتفي بمرشح قريب من الأفضل، وبالمقابل نريد التوظيف مرّة واحدة فقط. يجب أن نحترم شرطًا واحدًا للشركة: بعد كل مقابلة، يجب على الفور إما أن نعطي المكان الشاغر للمتقدّم أو نرفضه. ما التسوية بين تقليل عدد المقابلات ورفع مستوى المرشح المختار قدر الإمكان؟

يمكننا أن ننمذج هذه المسألة بالطريقة التالية: بمقدورنا بعد مقابلة كل متقدّم، أن نعطي لكل مرشح علامة، ولتكن $score(i)$ العلامة المحددة للمتقدّم i، ونفترض أنه لا يوجد متقدّمان يحصلان على العلامة نفسها. بعد مقابلة j متقدّمًا، نعرف أيّهم الأعلى علامة، ولكننا لا نعرف إذا كان أيٌّ من المتقدمين المتبقين $n - j$ سيحقق علامة أعلى. نقرّر اعتماد الاستراتيجيّة المتمثلة في اختيار عدد صحيح موجب $k < n$، نقابل ونرفض أول k مرشحًا، ونوظّف أول متقدّم يليهم يحقّق علامة أعلى من كل المرشحين السابقين. إذا تبيّن أن أفضل متقدّم كان بين المقابلين k الأوائل، فسنوظّف المتقدّم الأخير n. نصوغ هذه الاستراتيجية في الإجراء ON-LINE-MAXIMUM(k, n) الذي يعيد دليل المرشح الذي نريد أن نوظّفه.

```
ON-LINE-MAXIMUM(k, n)
1  bestscore = −∞
2  for i = 1 to k
3      if score(i) > bestscore
4          bestscore = score(i)
5  for i = k + 1 to n
6      if score(i) > bestscore
7          return i
8  return n
```

نريد أن نحدّد، لكل قيمة ممكنة لـ k، احتمال أن نوظّف أفضل متقدّم ثمّ سنختار أفضل k ممكنة، وننفذ الاستراتيجية باستخدام هذه القيمة. لنفترض حاليًّا أن k ثابتة، ليكن $M(j) = \max_{1 \le i \le j} \{score(i)\}$ أعلى علامة بين المتقدمين من 1 إلى j. ليكن S الحدث المتمثل في نجاحنا في اختيار أفضل مرشح، وليكن S_i

الحدث المتمثل في نجاحنا عندما يكون أفضلُ مرشحٍ هو المرشحَ ذا الترتيب i في المقابلات. لما كانت الأحداثُ المختلفةُ S_i منفصلةً، فإن $\Pr\{S\} = \sum_{i=1}^{n} \Pr\{S_i\}$. وإذا لاحظنا أننا لا ننجح أبدًا عندما يكون أفضل متقدّم من بين المتقدمين k الأوائل، فيكون لدينا $\Pr\{S_i\} = 0$ عندما $i = 1, 2, .., k$، إذن نحصل على

$$\Pr\{S\} = \sum_{i=k+1}^{n} \Pr\{S_i\} . \tag{12.5}$$

سنحسب الآن $\Pr\{S_i\}$. حتى ننجح عندما يكون أفضل متقدّم هو المتقدّم i، يجب أن يحدث أمران. أولاً، يجب أن يكون المتقدّم الأفضل في الموقع i، وهذا حدثٌ نسميه B_i. ثانيًا يجب ألاّ تختار الخوارزميةُ أيَّ متقدّم بين الموقعين $k+1$ و $i-1$، وهذا سيحدث فقط إذا وجدنا عندما يحقق j المتراجحة $k+1 \leq j \leq i-1$ أن $score(j) < bestscore$ في السطر 6. (لما كانت العلامات وحيدة، يمكننا تجاهل إمكان أن يكون $score(j) = bestscore$). بمعنى آخر، يجب أن تكون كل العلامات $score(k+1)$ حتى $score(i-1)$ أصغر من $M(k)$؛ إذا كان أيٌّ منها أكبرَ من $M(k)$، فإننا سنعيد إذن دليل أول علامة تفوق $M(k)$. نستخدم O_i لنشير إلى الحدث المتمثل في عدم اختيار أيٍّ من المتقدمين في المواقع من $k+1$ حتى $i-1$. إن الحدثَيْن B_i و O_i هما، لحسن الحظ، مستقلاّن. فالحدث O_i يتعلّق فقط بترتيب القيم في المواقع من 1 حتى $i-1$، على حين أن الحدث B_i يتعلّق فقط بكون القيمة في الموقع i أكبر من القيم في كل المواقع الأخرى. إن ترتيب القيم في المواقع من 1 إلى $i-1$ لا يؤثر على كون القيمة في الموقع i أكبر منها كلها، والقيمة في الموقع i لا تؤثر على ترتيب القيم في المواقع من 1 حتى $i-1$. وهكذا يمكننا تطبيق المعادلة (ت.15) لنحصل على

$$\Pr\{S_i\} = \Pr\{B_i \cap O_i\} = \Pr\{B_i\}\Pr\{O_i\} .$$

من الواضح أن الاحتمال $\Pr\{B_i\}$ هو $1/n$، لأن القيمة العظمى يمكن أن تكون في أي موقع من بين n موقعًا باحتمالٍ متساوٍ. ولكي يقع الحدث O_i يجب أن تكون القيمة العظمى للقيم في المواقع من 1 حتى $i-1$ في موقعٍ من بين المواقع k الأولى، وباحتمال ورودٍ متساوٍ في أي موقع منها بين هذه المواقع، التي عددها $i-1$. إذن $\Pr\{O_i\} = k/(i-1)$ و $\Pr\{S_i\} = k/(n(i-1))$. وباستخدام المعادلة (12.5). يكون لدينا

$$\begin{aligned}
\Pr\{S\} &= \sum_{i=k+1}^{n} \Pr\{S_i\} \\
&= \sum_{i=k+1}^{n} \frac{k}{n(i-1)} \\
&= \frac{k}{n} \sum_{i=k+1}^{n} \frac{1}{n-1}
\end{aligned}$$

$$= \frac{k}{n} \sum_{i=k}^{n-1} \frac{1}{i} \; .$$

وحتى نَحُدّ هذا المجموع من الأعلى ومن الأسفل، نقرّب باستخدام التكاملات. فمن المتراجحات (أ.12) يكون لدينا

$$\int_{k}^{n} \frac{1}{x} dx \leq \sum_{i=k}^{n-1} \frac{1}{i} \leq \int_{k-1}^{n-1} \frac{1}{x} dx \; .$$

إن حساب هذه التكاملات المعرّفة يعطينا الحدود

$$\frac{k}{n}(\ln n - \ln k) \leq \Pr\{S\} \leq \frac{k}{n}(\ln(n-1) - \ln(k-1)) \; ,$$

وهي حدود ملاصقة نسبيًّا لـ $\Pr\{S\}$. ولما كنا نريد أن نجعل احتمال النجاح أكبر ما يمكن، فإننا سنركّز على اختيار قيمة k التي تجعل الحدَّ الأدنى على $\Pr\{S\}$ أكبر ما يمكن. (إلى جانب أنه من الأسهل جعل عبارة الحد الأدنى أكبر ما يمكن مقارنة بتكبير عبارة الحدِّ الأعلى.) بمفاضلة العبارة $(k/n)(\ln n - \ln k)$ بالنسبة إلى k، نحصل على

$$\frac{1}{n}(\ln n - \ln k - 1)$$

وبجعل هذا المشتق معدومًا نرى أن الحد الأدنى على الاحتمال يصبح أكبر ما يمكن عندما يكون $\ln k = \ln n - 1 = \ln(n/e)$، أو بعبارة مكافئة $k = n/e$. إذن، إذا نفذّنا استراتيجتنا مع $k = n/e$، فسننجح في توظيف أفضل متقدّم باحتمال قدره $1/e$ على الأقل.

تمارين

1-4.5

ما عدد الأشخاص الذين يجب أن يكونوا موجدين في غرفة حتى يكون احتمال أن يكون هناك شخص يشاركك في يوم ميلادك، هو 1/2 على الأقل؟ ما عدد الأشخاص اللازم حتى يكون احتمال أن يوجد شخصان على الأقل وُلدا في 4 تموز، أكبر من 1/2؟

2-4.5

افترض أن كرات رُمِيَتْ في b سلة. كل رمية مستقلّة ويتساوى احتمال أن تنتهي أية كرة في أية سلة. ما هو عدد الرميات الكرات المتوقع قبل أن تحتوي على الأقل واحدة من السلال على كرتين؟

3-4.5 ⋆

هل من الضروري، عند دراسة متناقضة يوم الميلاد أن تكون أيام الميلاد مستقلة فيما بينها، أم أن الاستقلال الثنائي (زوجًا زوجًا) كافٍ؟ علِّل إجابتك.

★ *4-4.5*

كم مدعوًّا يجب أن تدعو إلى حفلة حتى يصبح من المحتمل اجتماع *ثلاثة* أشخاص يشتركون في يوم ميلادهم؟

★ *5-4.5*

ما احتمال أن تكون متتالية محرفية– k string– k على مجموعة من n محرف هي فعلاً تبديل– k؟ ما علاقة هذا السؤال بمتناقضة يوم الميلاد؟

★ *6-4.5*

افترض أننا رمينا n كرة في n سلة، حيث كل رمية مستقلّة، واحتمال أن تنتهي الكرة في أية علبة متساوٍ أيضًا. ما هو عدد السلال الفارغة المتوقع؟ ما هو العدد المتوقع للسلال التي تحتوي كرة واحدة؟

★ *7-4.5*

حَسِّن الحدّ الأدنى على طول ضربة الحظ، وذلك بأن تبيّن أنه، عند رمي قطعة نقد عادلة n رميةً، فهناك احتمال أقل من $1/n$ ألاّ تحدث ضربة حظ أطول من $\lg n - 2\lg\lg n$ وجهًا متتاليًا.

مسائل

1-5 العد الاحتمالي

بإمكاننا، باستخدام عدّادٍ ذي b بتًّا، أن نَعُدّ بالترتيب حتى $2^b - 1$ فقط. وباستخدام ***العدّ الاحتمالي probabilistic counting*** الذي ابتدعه موريس R. Morris، بإمكاننا أن نَعُدَّ حتى قيمة أعلى بكثير مقابل خسارة بعض الدقة.

نجعل قيمة عدّادٍ i يمثّلُ عدًّا لـ n_i لكل $i = 0, \dots, 2^b - 1$ حيث تشكل القيم n_i متتالية متزايدة من القيم الموجبة. نفترض أن القيمة البدائية للعدّاد هي 0، وهي تمثّل عدًّا لـ $n_0 = 0$. تعمل العملية INCREMENT على عدّاد يحتوي القيمة i على نحوٍ احتمالي. إذا كان $i = 2^b - 1$ فسينتج تقرير خطأ فيضٍ overflow error. وما سوى هذه الحالة، فإن العدّاد يزداد 1 باحتمال $1/(n_{i+1} - n_i)$، ويبقى على حاله باحتمال $1 - 1/(n_{i+1} - n_i)$.

إذا اخترنا $n_i = i$ لكل $i \geq 0$، فإن العدّاد يكون عدادًا عاديًّا ولكن تظهر حالات أكثر إثارة للاهتمام عندما نختار، $n_i = 2^{i-1}$ مثلاً، عندما $i > 0$ أو $n_i = F_i$ (عدد فيبوناتشي ذو الرقم i. انظر المقطع 2.3).

لنفترض، في هذه المسألة، أن القيمة n_{2^b-1} كبيرة كفاية بحيث يكون احتمال خطأ الفيض مهملاً.

أ. بيّن أن القيمة المتوقعة التي يعطيها العدّاد بعد n عملية INCREMENT هي n تمامًا.

ب. يعتمد تحليل تباين variance العد الممثّل بالعداد على المتتالية n_i. لنأخذ حالةً بسيطةً: $n_i = 100i$ لكل $i \geq 0$. قدِّرْ تباين القيمة الممثلة في العداد بعد n عملية INCREMENT.

2-5 البحث في صفيفة غير مفروزة

تدرس هذه المسألة ثلاث خوارزميات للبحث عن قيمة x في صفيفة غير مفروزة A مؤلّفة من n عنصرًا.

لندرس الاستراتيجية ذات العشوائية المضافة التالية: اختر دليلاً عشوائيًا i في A. إذا كان $A[i] = x$، ينتهي عملنا، وإلاّ فإننا نتابع البحث باختيار دليل عشوائي جديد في A. نتابع اختيار أدلّة عشوائية من A حتى نجد دليلاً j بحيث يكون $A[j] = x$ أو حتى نكون قد تحققنا من كل عنصر A. لاحظ أننا نختار الدليل من مجموعة الأدلة الكاملة في كل مرّة، ولهذا السبب قد نتفحّص عنصرًا ما أكثر من مرّة.

أ. اكتب شبه رمازٍ للإجراء RANDOM-SEARCH الذي ينجّز الاستراتيجيّة السابقة. تأكّد أن إجراءك يتوقف عندما تكون كل الأدلّة في A قد اختيرت.

ب. افترض أنّ هناك دليلاً واحدًا i بحيث يكون $A[i] = x$. ما هو العدد المتوقع للأدلة التي يجب اختيارها في A قبل العثور على x وتوقّف RANDOM-SEARCH؟

ت. بتعميم حلّك للسؤال (ب)، افترض أن هناك $k \geq 1$ دليلاً i بحيث يكون $A[i] = x$. ما هو العدد المتوقع للأدلة التي يجب اختيارها في A قبل العثور على x وتوقّف RANDOM-SEARCH؟ يجب أن يكون جوابك بدلالة n و k.

ث. افترض أنه لا يوجد أي دليل i بحيث يكون $A[i] = x$. ما هو العدد المتوقع للأدلّة التي يجب اختيارها في A حتى تكون كل الأدلة في A قد اختيرت وحتى يتوقف RANDOM-SEARCH؟

لندرسْ الآن خوارزمية بحث خطيّة حتميّة، نشير إليها بـ DETERMINISTIC-SEARCH. تقوم الخوارزمية، تحديدًا، بالبحث عن x داخل A بالترتيب، متفحصةً $A[1], A[2], A[3], \ldots, A[n]$ بالترتيب حتى نجد $A[i] = x$ أو حتى الوصول إلى نهاية الصفيفة. افترض أن كل التباديل الممكنة لصفيفة الدخل متساوية الاحتمال.

ج. افترض أنّ هناك دليلاً واحدًا i بحيث يكون $A[i] = x$. ما هو زمن التنفيذ المتوقع لـ DETERMINISTIC-SEARCH؟ وما هو زمن تنفيذ DETERMINISTIC-SEARCH في أسوأ الحالات؟

ح. بتعميم حلّك للسؤال (ج)، افترض أن هناك $k \geq 1$ دليلاً i بحيث يكون $A[i] = x$. ما هو زمن التنفيذ المتوقع لـ DETERMINISTIC-SEARCH؟ وما هو زمن تنفيذ DETERMINISTIC-SEARCH في أسوأ الحالات؟ يجب أن يكون جوابك بدلالة n و k.

خ. افترض أنه لا يوجد أي دليل i بحيث يكون $A[i] = x$. ما هو زمن التنفيذ المتوقع لـ DETERMINISTIC-SEARCH؟ وما هو زمن تنفيذ DETERMINISTIC-SEARCH في أسوأ الحالات؟

وأخيرًا، لندرس خوارزمية ذات عشوائية مضافة SCRAMBLE-SEARCH تعمل أولاً على خلط صفيفة الدخل عشوائيًا، ثمّ على تنفيذ البحث الخطيّ الحتميّ المعطى هنا على الصفيفة الناتجة.

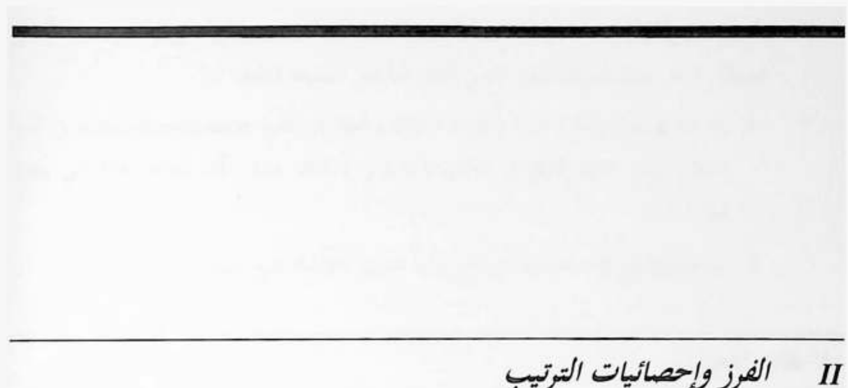
د. إذا كان k عدد الأدلة i بحيث يكون $A[i] = x$ ، أعطِ زمن تنفيذ SCRAMBLE-SEARCH في أسوأ الحالات وزمن التنفيذ المتوقع في الحالتين $k = 0$ و $k = 1$. عمّمْ حلّك ليعالج الحالة التي يكون فيها $k \geq 1$.

ذ. ما الخوارزمية التي قد تستخدمها من خوارزميات البحث الثلاث؟ اشرح جوابك.

ملاحظات الفصل

تضم المراجع Bollobás [54]، و Hofri [174] و Spencer [321] كمًّا كبيرًا من الطرق الاحتمالية المتقدمة. ويقدّم كلٌّ من Karp [200] و Rabin [288] مناقشةً لفوائد الخوارزميات ذات العشوائية المضافة ومعاينةً لها. ويقدّم الكتاب التدريسي Motwani و Raghavan [262] دراسة موسّعة للخوارزميات ذات العشوائية المضافة.

لقد جرت دراسة عدّة نماذج معدّلة من مسألة التوظيف على نطاق واسع. والأكثر شيوعًا أن يشار إلى هذه المسائل باسم "مسائل السكرتيرة secretary problems". يُعَدُّ مقال Ajtai و Meggido و Waarts [11] مثالاً على أعمالٍ في هذا المجال.



II الفرز وإحصائيات الترتيب

تمهيد

يعرض هذا الباب خوارزميات عديدة تحل مسألة الفرز التالية:

الدخل: متتالية من n عددًا $\langle a_1, a_2, \ldots, a_n \rangle$.

الخرج: تبديل (إعادة ترتيب) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ متتالية الدخل بحيث تحقِّق $\langle a'_1 \leq a'_2 \leq \cdots \leq a'_n \rangle$.

تكون متتالية الدخل عادةً صفيفةً من n عنصرًا، ويمكن تمثيلها بطرق أخرى مختلفة، كاللائحة المترابطة مثلاً.

بنية المعطيات

نادرًا ما تكون الأعداد التي نريد فرزها قيمًا مُنفصلةً عمليًّا، بل يكون كلٌّ منها، عادةً، جزءًا من تشكيلة من المعطيات تسمَّى ***تسجيلة record***. تتضمن كلُّ تسجيلةٍ ***مفتاحًا key***، هو القيمة التي يجب فرزها. وتتألف بقية التسجيلة من ***معطيات تابعة satellite data***، تُحرَّك عادةً مع المفتاح. عمليًّا، عندما تقوم خوارزميةُ الفرز بتبديل المفاتيح، فلا بد أن تُبدِّل هذه الخوارزميةُ المعطياتِ التابعةَ أيضًا. فإذا تضمَّنت كلُّ تسجيلةٍ حجمًا كبيرًا من المعطيات التابعة، فإننا غالبًا ما نبدِّل صفيفةً من مؤشرات التسجيلات بدلاً من التسجيلات نفسها، وذلك لتقليص حركة المعطيات.

إن تفاصيل التنجيز هذه هي التي تُميِّز في الحقيقة خوارزميةً ما من برنامجٍ متكامل. تَصِفُ خوارزمية الفرزِ ***الطريقةَ method*** التي تحدِّد بها الترتيب المفروز، بصرف النظر عن كوننا نفرز أعدادًا مستقلةً أم تسجيلات ضخمة تحوي كثيرًا من بايتات المعطيات التابعة. لذلك، عندما نركز على مسألة الفرز، فإننا نفترض نموذجيًّا بأن الدخل مؤلَّف من أعداد فقط. إن ترجمة خوارزمية فرز الأعداد إلى برنامجٍ لفرز التسجيلات هو أمر مباشر مفاهيميًّا (نظريًّا)، مع أنه في حالات هندسية محدَّدة، قد تجعلُ بعض التفاصيل الدقيقة الأخرى من مهمة البرمجة الفعلية تحديًا.

لماذا الفرز؟

يعتبرُ الكثير من علماء الحواسيب أن الفرزَ أهم المسائل الأساسية في دراسة الخوارزميات. وذلك لعدة أسباب:

- أحيانًا تكون الحاجة إلى فرز المعلومات أمرًا جوهريًّا في تطبيق ما. فمثلاً، تحتاج المصارف عند تجهيز بيانات الزبائن إلى فرز الشيكات وفق أرقامها.
- غالبًا ما تَستخدم الخوارزمياتُ الفرزَ باعتباره مساقًا فرعيًّا أساسيًّا. فمثلاً، قد يكون على البرنامج الذي يرسم أغراضًا بيانية متوضعة بعضها فوق بعض، أن يفرز هذه الأغراض وفق علاقة "فوق" بحيث يمكنه رسم هذه الأغراض من الأسفل إلى الأعلى. سنرى في هذا النص العديد من الخوارزميات التي تستخدم الفرزَ باعتباره مساقًا فرعيًّا.
- يمكننا أن نستنتج تشكيلة واسعة من خوارزميات الفرز، وهي تستخدم مجموعة غنية من التقنيات. والواقع، أن العديد من التقنيات الهامة المستخدَمة أثناء تصميم الخوارزمية تَظهر في متن خوارزمياتِ فرزٍ جرى تطويرها عبر السنين. ومن ثم، فالفرز مسألة ذات أهمية تاريخية أيضًا.
- يمكن أن نبرهن وجود حدٍّ أدنى غير بديهي للفرز (كما سنفعل في الفصل 8). تُطابق حدودنا العليا الفُضلى الحدَّ الأدنى على نحو مقارب، وبذلك نعلم أن خوارزمياتنا للفرز مُثلى على نحو مقارب. إضافة إلى ذلك، يمكننا استخدام الحد الأدنى للفرز لإثبات حدود دنيا لبعض المسائل الأخرى.
- تظهر العديد من المسائل الهندسية عند تنجيز خوارزميات الفرز. قد يعتمد أسرع برنامج فرزٍ خاص بحالة معينة على عدة عوامل، مثل المعرفة المسبقة عن المفاتيح والمعطيات التابعة، وبنيان الذاكرة (الذواكر المخبئة caches، والذاكرة الافتراضية virtual memory) للحاسوب المضيف، والبيئة البرمجية. تُعالَجُ العديدُ من هذه المسائل أمثليًّا على مستوى الخوارزميات، وليس بـ "إضافة تحسينات tweaking" إلى الرماز.

خوارزميات الفرز

قدمنا في الفصل الثاني خوارزميتَيْن لفرز n عددًا حقيقيًّا. أولاهما خوارزمية الفرز بالإدراج، وهي تتطلَّب زمنًا قدره $\Theta(n^2)$ في أسوأ الحالات. ولكن، لما كانت الحلقات الداخلية لهذه الخوارزمية محكمة، فهي خوارزمية فرزٍ سريعةٍ ***في المكان in-place*** في حال حجوم دخل صغيرة. (تذكَّر أن خوارزمية فرزٍ تفرز في المكان إذا كان عدد العناصر من صفيفة الدخل – التي تخزّن في أي وقت خارج الصفيفة – عددًا ثابتًا.) والثانية خوارزمية الفرز بالدمج، ولها زمن تنفيذ مقارب أفضل وهو $\Theta(n \lg n)$، ولكن إجراء MERGE الذي تستخدمه لا يُنَفَّذ في المكان.

سنقدم، في هذا الباب، خوارزميتَيْن إضافتين تفرزان أعدادًا حقيقية لا على التعيين. الأولى خوارزمية الفرز

بالكومة Heapsort، سنعرضها في الفصل 6. تفرز هذه الخوارزمية n عددًا في المكان في زمن $O(n\lg n)$، وتَستخدم بنيةَ معطيات هامة، تسمى كومة heap، يمكننا استخدامها أيضًا لتنجيز رتل ذو أولوية priority queue.

والثانية خوارزمية الفرز السريع quicksort، سنعرضها في الفصل 7. تَفرز هذه الخوارزمية n عددًا أيضًا في المكان، ولكن زمن تنفيذها في أسوأ الحالات هو $\Theta(n^2)$. ومع هذا فإن زمن تنفيذها المتوقع هو $\Theta(n\lg n)$، وعادة ما يفوق أداؤها عمليًّا خوارزمية الفرز بالكومة. ويمتاز رماز خوارزمية الفرز السريع بأنه محكم، كالفرز بالإدراج، لذلك فإن العامل الثابت المخفي في زمن تنفيذها صغير. وهي خوارزمية شائعة الاستخدام لفرز صفيفاتِ دخلٍ كبيرة.

وتُعَدُّ خوارزمياتُ الفرز بالإدراج، والفرز بالدمج، والفرز بالكومة، والفرز السريع خوارزمياتِ فرزٍ بالمقارنة comparison sorts، أي إنها تحدِّد الترتيب المفروز لصفيفة دخل بمقارنة عناصرها. يبدأ الفصل 8 بتقديم نموذج شجرة القرار decision-tree بهدف دراسة حدود أداء خوارزميات الفرز بالمقارنة. نبرهن، باستخدام هذا النموذج، وجود حد أدنى $\Omega(n\lg n)$ لزمن تنفيذ أي فرز بالمقارنة على n دخلاً في أسوأ الحالات، موضّحين بذلك أن الفرز بالكومة والفرز بالدمج هما خوارزميتا فرز بالمقارنة أمثليتان على نحو مقارب.

يتابع الفصل 8 بعد ذلك ليثبت أنه يمكننا التغلُّب على الحد الأدنى $\Omega(n\lg n)$ إذا استطعنا جَمْع معلوماتٍ عن ترتيب الدخل المفروز بأسلوبٍ مختلف عن مقارنة العناصر. فمثلاً، تفترض خوارزمية الفرز بالعد أن أعداد الدخل تقع ضمن المجموعة $\{0,1,\dots,k\}$. وباستخدام دليل الصفيفة أداةً لتحديد الترتيب النسبي، يستطيع الفرزُ بالعد فرزَ n عددًا في زمن $\Theta(k+n)$. ومن ثم، في حال $k=O(n)$ يكون زمن تنفيذ الفرز بالعد خطيًّا بالنسبة إلى طول صفيفة الدخل. يمكن استخدام خوارزمية ذات صلة، وهي الفرز حسب الأساس radix sort، لتوسيع مجال الفرز بالعد. فإذا كان علينا فرز n عددًا صحيحًا، وكل عدد فيه d رقمًا، وكل رقم يمكن أن يأخذ قيمًا محتملة قد تصل إلى k قيمةً على الأكثر، فيمكن للفرز حسب الأساس أن يفرز الأعداد في زمن $\Theta(d(n+k))$. وعندما يكون d ثابت و k من رتبة $O(n)$، فإن الفرز حسب الأساس يُنَفَّذ في زمن خطي. ثمة خوارزميةٌ ثالثة، هي الفرز بالدلاء bucket sort، وتتطلب معرفةً عن التوزع الاحتمالي للأعداد في صفيفة الدخل. يمكن لهذه الخوارزمية أن تفرز n عددًا حقيقيًا موزعًا بانتظام uniformly في المجال نصف المفتوح $[0,1)$ في زمن $O(n)$ في الحالة الوسطى.

يلخص الجدول التالي أزمان تنفيذ خوارزميات الفرز الموجودة في الفصل 2 والفصول من 6 إلى 8. تمثِّلُ n، كالمعتاد، عدد العناصر التي ستُفرَز. ففي حالة الفرز بالعد، تكون العناصر التي ستُفرَز أعدادًا صحيحة من المجموعة $\{0,1,\dots,k\}$. وفي حالة الفرز حسب الأساس، كل عنصر هو عدد من d رقمًا، حيث يأخذ كل رقمٍ k قيمة محتملة. وفي حالة الفرز بالدلاء، نفترض أن المفاتيح هي أعداد حقيقية موزعة بانتظام ضمن المجال نصف المفتوح $[0,1)$. يعطي العمود الأيسر زمن التنفيذ في الحالة الوسطى أو المتوقع، مبيِّنًا الحالةَ التي تعطيه

عندما يكون مغايرًا لزمن التنفيذ في أسوأ الحالات. لم نورد زمن التنفيذ في الحالة الوسطى للفرز بالكومة لأننا لم نحلله في هذا الكتاب.

الخوارزمية	زمن التنفيذ في أسوأ الحالات	زمن التنفيذ في الحالة الوسطى/المتوقع
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	—
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (المتوقع)
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k)$	$\Theta(d(n + k)$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (الحالة الوسطى)

إحصائيات الترتيب

إن إحصائية الترتيب من الرتبة i لمجموعة من n عددًا هي العدد ذو الترتيب i من حيث الصغر في المجموعة. يمكن طبعًا اختيار إحصائية الترتيب من الرتبة i بفرز الدخل وفهرسة العنصر ذي الترتيب i من الخرج. فإذا لم توجد أية فرضيات على توزع الدخل، فإن هذه الطريقة تُنَفَّذ في زمن $\Omega(n \lg n)$، وهو الحد الأدنى المبرهن عليه في الفصل 8.

نبيِّن في الفصل 9، أن بإمكاننا إيجاد العنصر ذي الترتيب i من حيث الصغر في زمن $O(n)$، ولو كانت العناصر أعدادًا حقيقية لا على التعيين. نقدم خوارزميةً ذات عشوائية مضافة بشبه رماز محكم يُنفَّذ في زمن $\Theta(n^2)$ في أسوأ الحالات، ولكن في زمن متوقع $O(n)$. نقدم أيضًا خوارزمية أعقد تُنَفَّذ في زمن $O(n)$ في أسوأ الحالات.

خلفية

مع أن أغلب هذا الباب لا يعتمد على الرياضيات المعقدة، إلا أن بعض المقاطع تتطلب درايةً رياضية. وبوجهٍ خاص، فإن تحليل الفرز السريع والفرز بالدلاء وخوارزمية إحصاء الترتيب يَستخدم الاحتمالات، التي عرضناها في الملحق ت، إضافة إلى المادة المتعلقة بالتحليل الاحتمالي والخوارزميات ذات العشوائية المضافة في الفصل 5. يتطلب تحليل خوارزمية إحصائيات الترتيب، الخطية الزمن في أسوأ الحالات، رياضيات أكثر تعقيدًا من الرياضيات اللازمة لتحليل أسوأ حالات لخوارزميات أخرى في هذا الباب.

6 الفرز بالكومة

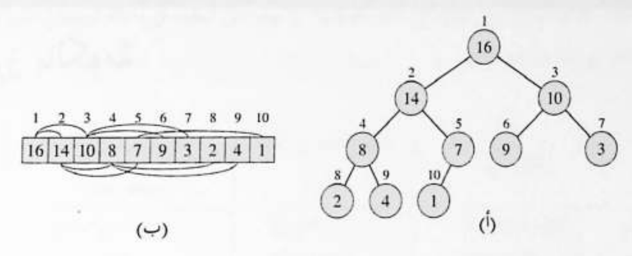
نعرض في هذا الفصل خوارزميةَ فرزٍ أخرى: الفرز بالكومة heapsort. وزمن تنفيذ خوارزمية هذا الفرز يماثل زمن تنفيذ الفرز بالدمج، ويساوي $O(n \lg n)$، وهو يخالف زمن تنفيذ الفرز بالإدراج. تفرز خوارزمية الفرز بالكومة في المكان، كالفرز بالإدراج وخلافًا للفرز بالدمج: إذ إنه يُخزِّنُ – في أي وقت – عددًا ثابتًا فقط من عناصر الصفيفة خارج صفيفة الدخل. وهكذا، تَجمعُ خوارزمية الفرز بالكومة أفضل واصفات خوارزميتَي الفرز اللتين ناقشناهما سابقًا.

يقدم الفرز بالكومة أيضًا تقنيةً أخرى لتصميم الخوارزميات: وهي أنه يَستخدم بنيةَ معطيات، نسميها في هذه الحالة "كومة heap"، لإدارة المعلومات. ولا تقتصر الفائدةُ من بنية معطيات "الكومة" على الفرز بالكومة فحسب، بل تتعداها لتشكيل رتلٍ ذي أولوية فعّال. ستظهر بنية المعطيات "الكومة" مجدَّدًا في خوارزمياتٍ تُعرَض في فصول لاحقة.

صِيغَ المصطلح "كومة" في الأصل في سياق الفرز بالكومة، لكنه أصبح يشير إلى "تخزين النفايات المجمعة" garbage-collected storage، كالذي توفره لغتا البرمجة Lisp و Java. لكن بنية المعطيات "الكومة" *ليست* تخزينًا للنفايات المجمعة، وحيثما نشير إلى الكومات في هذا الكتاب، فإننا نعني بنية المعطيات، وليس سمة لتجميع النفايات.

1.6 الكومات

بنيةُ المعطيات ***الكومة heap*** ***(الثنائية binary)*** هي غرضٌ صفيفي يمكن اعتباره شجرة ثنائية كاملة تقريبًا (انظر المقطع ب-3.5)، كما هو مبين في الشكل 6-1. فكلُّ عقدةٍ من الشجرة توافق عنصرًا من الصفيفة. وجميع مستويات الشجرة ممتلئة، ربما باستثناء المستوى الأدنى، الذي يُملأ ابتداءً من اليسار وحتى نقطةٍ ما. إنّ الصفيفة A التي تُمثِّل كومةً ما هي غرضٌ له واصفتان: $A.length$، التي تعطي (كالعادة) عددَ العناصر في الصفيفة، و $A.heap\text{-}size$، التي تمثِّل عددَ عناصر الكومة المخزَّنة في الصفيفة A. أي إن – بافتراض أن $A[1..A.length]$ يمكن أن تتضمن أعدادًا – العناصر الموجودة في $A[1..A.heap\text{-}size]$، حيث



الشكل 1.6 الكومة وفق الأكبر باعتبارها (أ) شجرة ثنائية و (ب) صفيفة. إن العدد الموجود ضمن الدائرة في كل عقدة من الشجرة هو القيمة المخزنة في تلك العقدة. والعدد الموجود في أعلى عقدة ما هو الدليل الموافق في الصفيفة. تبيّن الخطوطُ الموجودة أسفل وأعلى الصفيفة العلاقات من نمط أب–ابن؛ يكون الآباء دومًا إلى يسار أبنائهم. ارتفاع هذه الشجرة ثلاثة، وارتفاع العقدة التي دليلها 4 (وقيمتها 8) هو واحد.

$0 \le A.heap\text{-}size \le A.length$، هي وحدها العناصر التي يمكن أن تكون في الكومة. إن جذر الشجرة هو $A[1]$، وإذا أُعطينا دليلَ عقدة ما i، أمكننا بسهولة حساب أدلة الأب PARENT(i)، والابن الأيسر LEFT(i)، والابن الأيمن RIGHT(i) لهذه العقدة:

```
PARENT(i)
1  return ⌊i/2⌋

LEFT(i)
1  return 2i

RIGHT(i)
1  return 2i + 1
```

يمكن للإجراء LEFT حساب $2i$ بتعليمة واحدة، في أغلب الحواسيب، وذلك بإزاحة تمثيل i الاثنائي بتًّا واحدًا نحو اليسار. وبالمثل، يمكن للإجراء RIGHT حساب $2i + 1$ بسرعة، وذلك بإزاحة تمثيل i الاثنائي بتًّا واحدًا نحو اليسار، ثم وضع 1 في البت ذي المرتبة الدنيا. ويمكن للإجراء PARENT حساب $\lfloor i/2 \rfloor$ بإزاحة i بتًّا واحدًا نحو اليمين. وغالبًا ما تنجَز هذه الإجراءات الثلاثة، في التنجيز الجيد للفرز بالكومة، باعتبارها إجراءات "ماكرو" macros أو إجراءات "مُضمَّنة in-line".

ثمة نوعان من الكومات الثنائية: الكومة وفق الأكبر max-heap، والكومة وفق الأصغر min-heap. وفي كليهما تحقِّق القيمُ في العقد ***خاصيةَ الكومة heap property***، أي المميزات التي يعتمد عليها نمط الكومة. ***فخاصية الكومة وفق الأكبر max-heap property*** هي أن كل عقدة i عدا الجذر تحقق:

$$A[\text{PARENT}(i)] \ge A[i] ,$$

أي إن قيمة عقدةٍ ما تساوي على الأكثر قيمةَ أبيها. وبذلك، يُخزَّن العنصرُ الأكبر في كومة وفق الأكبر في الجذر، والشجرة الفرعية – التي جذرها عقدة ما – لا تتضمن قيمًا أقل أو تساوي تلك الموجودة في العقدة نفسها. أما ***الكومة وفق الأصغر min-heap***، فتنظَّم بالطريقة المعاكسة؛ ***فخاصية الكومة وفق الأصغر min-heap property*** هي أنه كل عقدة i عدا الجذر تحقِّق:

$A[\text{PARENT}(i)] \leq A[i]$.

ويكون أصغرُ عنصرٍ في كومةٍ وفق الأصغر موجودًا في جذرها.

نستخدم الكومات وفق الأكبر في خوارزمية الفرز بالكومة. تُنجِّز الكومات وفق الأصغر عادةً الأرتال ذات الأولويات، التي نناقشها في المقطع 5.6. وسنحدِّد بدقةٍ – في أيِّ تطبيق معيَّن –: هل نحن بحاجةٍ إلى كومةٍ وفق الأكبر أم إلى كومةٍ وفق الأصغر؟ فإذا كانت الخاصيات تنطبق على الكومة وفق الأكبر أو على الكومة وفق الأصغر، فإننا نستخدم المصطلح "كومة" فقط.

بالنظر إلى الكومة على أنها شجرة، نعرِّف ***ارتفاعَ height*** عقدةٍ في كومة بأنه عدد الوصلات edges الموجودة على أطول مسار بسيط نازل من العقدة إلى ورقةٍ ما، ونعرِّف ارتفاعَ الكومة بأنه ارتفاع جذرها. ولما كانت الكومة المؤلفة من n عنصرًا مهيكلة على أساس شجرة ثنائية كاملة، فإن ارتفاعها هو $\Theta(\lg n)$ (انظر التمرين 6.1-2). سنرى أن العمليات الأساسية على الكومات تُنَفَّذُ في زمن متناسبٍ طردًا مع ارتفاع الشجرة على الأكثر، وبذلك فهي تستغرق زمنًا $O(\lg n)$. يقدِّم ما تبقَّى من هذا الفصل عدة إجراءات أساسية ويبيِّن كيفية استخدامها في خوارزمية الفرز وفي بنية المعطيات ذات الأولوية.

- إن إجراء MAX-HEAPIFY، الذي يُنفَّذُ في زمن $O(\lg n)$، هو الأساس للمحافظة على خاصية الكومة وفق الأكبر.
- يولِّد إجراءُ BUILD-MAX-HEAP، الذي يُنفَّذُ في زمنٍ خطِّي، كومةً وفق الأكبر من صفيفة دخلٍ غير مرتبة.
- يَفْرز إجراءُ HEAPSORT، الذي يُنفَّذُ في زمن $O(n \lg n)$، صفيفةً في المكان.
- تسمح الإجراءات MAX-HEAP-INSERT و HEAP-EXTRACT-MAX و HEAP-INCREASE-KEY و HEAP-MAXIMUM، التي تنفَّذ في زمن $O(\lg n)$، لبنية المعطيات بتنجيز رتلٍ ذي أولوية.

تمارين

1-1.6

ما هو عدد العناصر الأصغري والأعظمي في كومةٍ ارتفاعها h؟

2-1.6

بيّن أن ارتفاع كومةٍ فيها n عنصرًا هو $\lfloor \lg n \rfloor$.

3-1.6

بيّن أنه في أية شجرةٍ فرعيةٍ لكومةٍ وفق الأكبر، يتضمن جذرُ الشجرة الفرعية القيمةَ العظمى للعناصر الموجودة في أي مكان في هذه الشجرة الفرعية.

4-1.6

أين يمكن أن يوجد أصغر عنصر في كومة وفق الأكبر، بافتراض أن جميع العناصر متمايزة؟

5-1.6

هل تشكل صفيفةٌ ذاتُ ترتيبٍ مفروزٍ كومةً وفق الأصغر؟

6-1.6

هل تشكل الصفيفة التي قيمها ⟨23, 17, 14, 6, 13, 10, 1, 5, 7, 12⟩ كومةً وفق الأكبر؟

7-1.6

بيِّن أنه، عند استخدام التمثيل الصفيفي لفرز كومة ذات n عنصرًا، تكون الأوراق هي العقد التي دلائلها $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

2.6 الحفاظ على خاصية الكومة

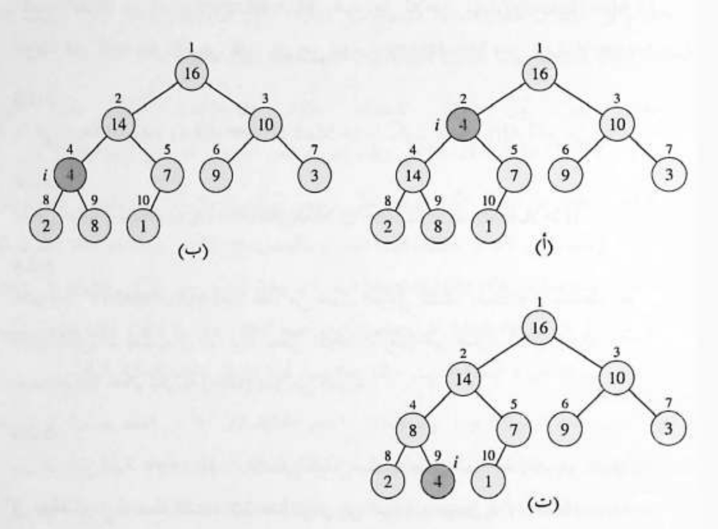
للحفاظ على خاصية الكومة وفق الأكبر، نستدعي الإجراء MAX-HEAPIFY. مُدخلاته هي صفيفة A ودليل i في هذه الصفيفة. يفترض MAX-HEAPIFY عند استدعائه أن الشجرتَيْن الثنائيتين ذواتَي الجذرين LEFT(i) و RIGHT(i) هما كومتان وفق الأكبر، ولكن قيمة $A[i]$ يمكن أن تكون أصغر من قيمة ابنيها، وبذلك فهي تخرق خاصية الكومة وفق الأكبر. يجعل MAX-HEAPIFY قيمةَ $A[i]$ "تغوص float down" في الكومة وفق الأكبر بحيث تستجيب الشجرة الفرعية التي جذرها عند الدليل i لخاصية الكومة وفق الأكبر.

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4       largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7       largest = r
 8  if largest ≠ i
 9       exchange A[i] with A[largest]
10       MAX-HEAPIFY(A, largest)
```

يبين الشكل 2.6 كيفية عمل MAX-HEAPIFY. يجري في كل مرحلة، تحديد العنصر الأكبر لـ $A[i]$

و $A[\text{LEFT}(i)]$ و $A[\text{RIGHT}(i)]$ وتخزين دليله في $largest$. إذا كان العنصر الأكبر هو $A[i]$، تكون الشجرة الفرعية التي جذرها عند العقدة i أصلاً كومة وفق الأكبر وينتهي الإجراء. وإلا، فإن أحد الابنين يتضمن العنصر الأكبر، وتجري مبادلة $A[i]$ مع $A[largest]$، وهذا يؤدي إلى تحقيق العقدة i وابنيها خاصية الكومة وفق الأكبر. لكن الآن أصبحت العقدة التي دليلها $largest$ تحتوي القيمة الأصلية $A[i]$، ومن ثم فإن الشجرة الفرعية التي جذرها $largest$ يمكن أن تخرق خاصية الكومة وفق الأكبر. وبالنتيجة، لا بد من استدعاء MAX-HEAPIFY عوديًا على هذه الشجرة الفرعية.

إن زمن تنفيذ MAX-HEAPIFY على شجرة فرعية – حجمها n وجذرها عقدة معطاة i – هو الزمن $\Theta(1)$ اللازم لتصحيح العلاقات بين العناصر $A[i]$ و $A[\text{LEFT}(i)]$ و $A[\text{RIGHT}(i)]$، إضافة إلى الزمن اللازم لتنفيذ MAX-HEAPIFY على شجرة فرعية جذرها أحد أبناء العقدة i (بافتراض حدوث الاستدعاء العودي). حجم كل من الشجرتين الفرعيتين للابنين لا يتجاوز $2n/3$ على الأكثر– تحدث أسوأ الحالات



الشكل 2.6 كيفية عمل MAX-HEAPIFY$(A,2)$، في حال $A.heap\text{-}size = 10$. (أ) التشكيلة الابتدائية، حيث تخرق $A[2]$ عند العقدة $i = 2$ خاصية الكومة وفق الأكبر، لأنها ليست أكبر من أيٍّ من ابنيها. تُستعاد في **(ب)** خاصية الكومة وفق الأكبر في العقدة 2، وذلك بمبادلة $A[2]$ بـ $A[4]$ التي تؤدي إلى خرق خاصية الكومة وفق الأكبر للعقدة 4. لدينا الآن $i = 4$ عند الاستدعاء العودي MAX-HEAPIFY$(A,4)$. وبعد مبادلة $A[4]$ بـ $A[9]$، كما هو مبين في **(ت)**، يجري تصحيح العقدة 4، ولا يؤدي الاستدعاء العودي MAX-HEAPIFY$(A,9)$ إلى حدوث تغييرات إضافية في بنية المعطيات.

عندما يكون نصف المستوى الأدنى من الشجرة ممتلئ تمامًا — ومن ثم يمكن وصف زمن تنفيذ MAX-HEAPIFY بالمعادلة التكرارية

$$T(n) \le T(2n/3) + \Theta(1) .$$

يكون حل هذه المعادلة التكرارية، حسب الحالة الثانية من النظرية العامة (النظرية 1.4)، هو $T(n) = O(\lg n)$. وبالمقابل، يمكننا وصف زمن تنفيذ MAX-HEAPIFY على عقدة ارتفاعها h بأنه $O(h)$.

تمارين

1-2.6

وضّح، باستخدام الشكل 2.6 نموذجًا، كيفية تطبيق MAX-HEAPIFY$(A, 3)$ على الصفيفة

$A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$

2-2.6

اكتب، انطلاقًا من إجراء MAX-HEAPIFY، شبه رماز للإجراء MIN-HEAPIFY(A, i) الذي يُنجز العملية الموافقة على كومة وفق الأصغر. قارن بين زمن تنفيذ MIN-HEAPIFY وزمن تنفيذ MAX-HEAPIFY.

3-2.6

ما هي نتيجة استدعاء MAX-HEAPIFY(A, i) عندما يكون العنصر $A[i]$ أكبر من أبنائه؟

4-2.6

ما هي نتيجة استدعاء MAX-HEAPIFY(A, i) في حالة $i > A.heap\text{-}size/2$؟

5-2.6

يعتبر رماز MAX-HEAPIFY فعالاً جدًّا من حيث العوامل الثابتة، باستثناء الاستدعاء العودي في السطر 10، الذي يمكن أن يسبِّب في أن تولِّد بعضُ المترجمات رمازًا غير فعال. اكتب إجراء MAX-HEAPIFY فعالاً يَستخدم بنية تحكم تكرارية (حلقة) بدلاً من العودية.

6-2.6

بيّن أن زمن تنفيذ MAX-HEAPIFY على كومة حجمها n في أسوأ الحالات هو $\Omega(\lg n)$. (*تلميح*: أعطِ، في حالة كومةٍ ذات n عقدة، قيم عقد تؤدي إلى استدعاءٍ عوديٍّ لـ MAX-HEAPIFY عند كل عقدة من مسار بسيط ابتداءً من الجذر نزولاً إلى ورقة.)

3.6 بناء كومة

يمكننا استخدام الإجراء MAX-HEAPIFY بطريقة صعودية لتحويل صفيفة $A[1 \,..\, n]$، حيث $n = A.length$، إلى كومة وفق الأكبر. استنادًا إلى التمرين 7-1.6، فإن العناصر في الصفيفة الجزئية

$A[(\lfloor n/2 \rfloor + 1)\,..\,n]$ كلُّها أوراق الشجرة، وبذلك فكلٌّ منها يشكل كومةً ذات عنصر واحد يمكن البدء به. يفحص إجراء BUILD-MAX-HEAP بقية العقد في الشجرة وينفذ MAX-HEAPIFY على كلٍّ منها.

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3       MAX-HEAPIFY(A, i)
```

يبين الشكل 3.6 مثالاً على عمل BUILD-MAX-HEAP.

لبيان لماذا يعمل BUILD-MAX-HEAP بصورة صحيحة، نستخدم لامتغير الحلقة:

في بداية كل تكرار من حلقة **for** في الأسطر 2-3، تكون كلُّ عقدةٍ من العقد $i+1, i+2, \ldots, n$ هي الجذر لكومةٍ وفق الأكبر.

نحتاج إلى بيان أن هذا اللامتغير صحيحٌ قبل التكرار الأول للحلقة، وأن كل تكرار للحلقة يحافظ على اللامتغير، وأن اللامتغير يوفر خاصية مفيدة لبيان الصحة عند انتهاء الحلقة.

الاستبداء: قبل أول تكرار للحلقة، يكون $i = \lfloor n/2 \rfloor$. وكلُّ عقدةٍ من العقد $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ هي ورقة، ومن ثَم فمن البديهي أنها جذرُ كومةٍ وفق الأكبر.

المحافظة: للتحقق من أن كل تكرار يحافظ على لامتغير الحلقة، لاحظ أن ابني العقدة i مرقمان بأعداد أكبر من i. واستنادًا إلى لامتغير الحلقة، فهما جذران لكومتين وفق الأكبر. وهذا هو تمامًا الشرط اللازم لكي يجعل الاستدعاءُ $\text{MAX-HEAPIFY}(A, i)$ العقدةَ i جذرًا لكومةٍ وفق الأكبر. إضافة إلى ذلك، يحافظ الاستدعاءُ MAX-HEAPIFY على خاصية كون جميع العقد $i+1, i+2, \ldots n$، جذورًا لكوماتٍ وفق الأكبر. إن إنقاص i ضمن تحديث حلقة **for** يعيد تهيئة لامتغير الحلقة للتكرار التالي.

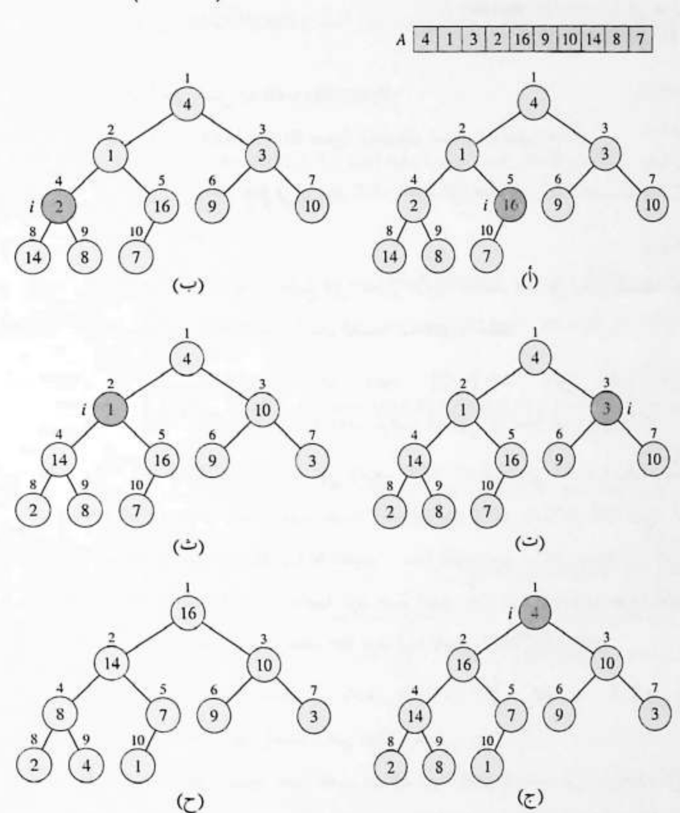
الانتهاء: لدينا عند النهاية $i = 0$. واستنادًا إلى لامتغير الحلقة، فإن كلاً من العقد $1, 2, \ldots n$ هي جذر كومةٍ وفق الأكبر. وبوجهٍ خاص، فإن العقدة 1 هي كذلك أيضًا.

يمكننا حساب حدٍّ أعلى بسيط لزمن تنفيذ BUILD-MAX-HEAP كما يلي: يستغرق كلُّ استدعاء لإجراء MAX-HEAPIFY زمنًا $O(\lg n)$، ويقوم BUILD-MAX-HEAP بـ $O(n)$ استدعاءً مماثلاً. لذلك، فإن زمن التنفيذ هو $O(n \lg n)$. ومع أن هذا الحد الأعلى صحيح، إلا أنه ليس مُحكمًا تقاربيًّا.

يمكننا استنتاج حدٍّ أكثر إحكامًا بملاحظة أن الزمن اللازم لتنفيذ MAX-HEAPIFY على عقدة يتغيَّر بتغيُّر ارتفاع العقدة في الشجرة، وأن ارتفاعات أغلب العقد صغيرة. يعتمد تحليلنا الأكثر إحكامًا على خاصية أن ارتفاع كومة ذات n عنصرًا هو $\lfloor \lg n \rfloor$ (انظر التمرين 1.6-2)، وأن فيها $\lceil n/2^{h+1} \rceil$ عقدة على الأكثر ارتفاعها h (انظر التمرين 3.6-3).

إن الزمن اللازم لتنفيذ MAX-HEAPIFY عند استدعائه على عقدة ارتفاعها h هو $O(h)$، وبذلك يمكننا التعبير عن أن الكلفة الكلية لإجراء BUILD-MAX-HEAP محدودةٌ من الأعلى كما يلي:

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) .$$



الشكل 3.6 كيفية عمل BUILD-MAX-HEAP، حيث تُظهر بنية المعطيات قبل استدعاء MAX-HEAPIFY في السطر 3 من BUILD-MAX-HEAP. (أ) صفيفة دخل A فيها 10 عناصر والشجرة الثنائية التي تمثلها. يبين الشكل أن دليل الحلقة i يشير إلى العقدة 5 قبل استدعاء MAX-HEAPIFY(A, i). **(ب)** بنية المعطيات الناتجة. يشير دليل الحلقة i في التكرار التالي إلى العقدة 4. **(ت)-(ج)** التكرارات اللاحقة لحلقة **for** في BUILD-MAX-HEAP. لاحظ أنه عند كل استدعاء لإجرائية MAX-HEAPIFY على عقدة، تكون الشجرتان الفرعيتان لهذه العقدة كلتاهما كومةً وفق الأكبر. (ح) الكومة وفق الأكبر بعد انتهاء BUILD-MAX-HEAP.

نقيّم المجموع الأخير بتعويض $x = 1/2$ في الصيغة (أ.8)، فينتج

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2}$$
$$= 2 \, .$$

وبذلك، يمكن وضع حدٍّ لزمن تنفيذ BUILD-MAX-HEAP كما يلي

$$O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right)$$
$$= O(n) \, .$$

ومن ثم، يمكننا بناء كومةٍ وفق الأكبر انطلاقًا من صفيفةٍ غير مرتبة في زمن خطي.

يمكننا بناء كومة وفق الأصغر باستخدام الإجراء BUILD-MIN-HEAP، الذي يشبه BUILD-MAX-HEAP مع الاستعاضة عن استدعاء MAX-HEAPIFY في السطر 3 باستدعاء MIN-HEAPIFY (انظر التمرين 6.2-2). يولِّد BUILD-MIN-HEAP كومةً وفق الأصغر من صفيفةٍ خطيةٍ غير مرتبة في زمن خطي.

تمارين

6.3-1

وضِّح، باستخدام الشكل 6.3 نموذجًا، عَمَلَ BUILD-MAX-HEAP على الصفيفة

$A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$

6.3-2

لماذا نرغب في أن يتناقص دليل الحلقة i في السطر 2 من BUILD-MAX-HEAP من $\lfloor A.length/2 \rfloor$ إلى 1 بدلاً من أن يتزايد من 1 إلى $\lfloor A.length/2 \rfloor$؟

6.3-3

بيِّن أنه يوجد على الأكثر $\lceil n/2^{h+1} \rceil$ عقدةً ارتفاعها h، في أية كومة ذات n عنصرًا.

6.4 خوارزمية الفرز بالكومة

تبدأ خوارزمية الفرز بالكومة باستخدام BUILD-MAX-HEAP لبناء كومة وفق الأكبر من صفيفة دخل $A[1 \,..\, n]$، حيث $n = A.length$. ولما كان العنصر الأكبر في الصفيفة مخزنًا في الجذر $A[1]$، فيمكن وضعه في موقعه النهائي الصحيح بتبديله بـ $A[n]$. إذا تجاهلنا الآن العقدة n من الكومة – ويمكننا فِعْل ذلك ببساطة بتقليص $A.heap\text{-}size$ – نلاحظ أن أبناء الجذر تبقى كوماتٍ وفق الأكبر، ولكن يمكن لعنصر

الجذر الجديد أن يخرق خاصية الكومة وفق الأكبر. ومع ذلك، فإن كل ما نحتاج إليه لاستعادة خاصية الكومة وفق الأكبر هو استدعاء إجراء MAX-HEAPIFY$(A,1)$، الذي يُبقِي في $A[1\,..\,n-1]$ كومةً وفق الأكبر. بعد ذلك، تعيد خوارزميةُ الفرز بالكومة هذه الإجرائية على الكومة وفق الأكبر التي حجمها $n-1$ لتصل إلى كومة حجمها 2. (انظر التمرين 6.4-2 لتحديد لامتغير الحلقة.)

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for A.i = A.length downto 2
3       exchange A[1] with A[i]
4       A.heap-size = A.heap-size − 1
5       MAX-HEAPIFY(A, 1)
```

يعرض الشكل 6-4 مثالاً لتطبيق HEAPSORT بعد أن يقوم السطر 1 ببناء الكومة وفق الأكبر الأولية. ويبيِّن هذا الشكلُ الكومةَ وفق الأكبر قبل إجراء أول تكرار لحلقة **for** في الأسطر 2-5 وبعد كل تكرار.

تستغرق إجرائية HEAPSORT زمنًا $O(n\lg n)$، لأن استدعاء BUILD-MAX-HEAP يستغرق زمنًا $O(n)$، ويستغرق كلٌّ من الـ $n-1$ استدعاءً لإجراء MAX-HEAPIFY زمنًا $O(\lg n)$.

تمارين

1-4.6

وضِّح، باستخدام الشكل 4.6 نموذجًا، عَمَلَ HEAPSORT على الصفيفة

$A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$

2-4.6

برهنْ صحةَ HEAPSORT باستخدام لامتغيِّر الحلقة التالي:

في بداية كل تكرار لحلقة **for** في الأسطر 2-5، تكون الصفيفة الجزئية $A[1\,..\,i]$ كومةً وفق الأكبر حاويةً أصغرَ i عنصرًا من $A[1\,..\,n]$، وتتضمن الصفيفة الجزئية $A[i+1\,..\,n]$ أكبرَ $n-i$ عنصرٍ من $A[1\,..\,n]$ مرتبة.

3-4.6

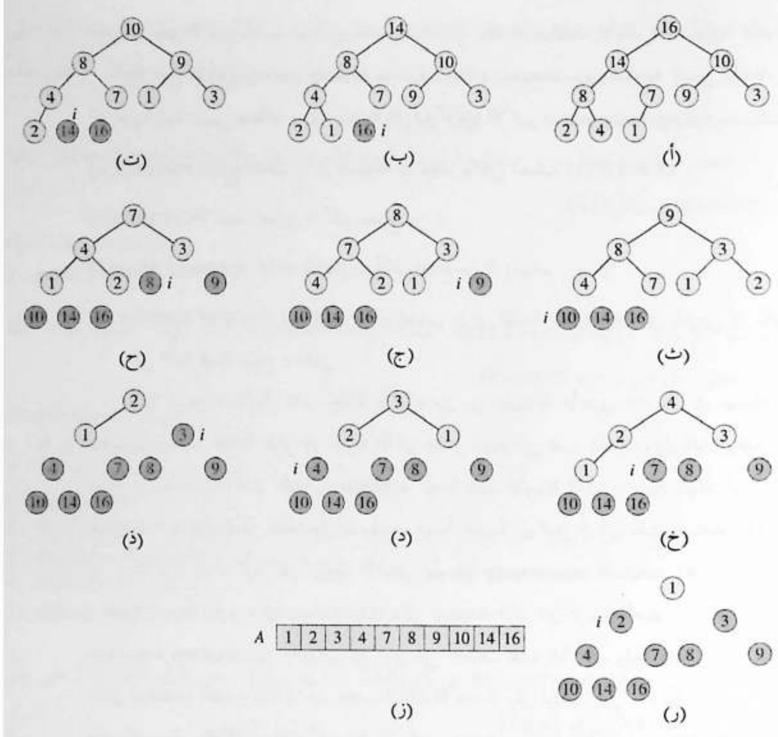
ما زمن تنفيذ الفرز بالكومة على صفيفة A طولها n مرتبة أصلاً ترتيبًا متزايدًا؟ ماذا عن الترتيب المتناقص؟

4-4.6

بيِّن أن زمن تنفيذ HEAPSORT في أسوأ الحالات هو $\Omega(n\lg n)$.

⋆ *5-4.6*

بين أنه عندما تكون جميع العناصر متمايزة، يكون زمن تنفيذ HEAPSORT بأحسن الحالات هو $\Omega(n\lg n)$.



الشكل 4.6 كيفية عمل الفرز بالكومة. (أ) بنية المعطيات الكومة وفق الأكبر مباشرة بعد أن بناها BUILD-MAX-HEAP في السطر 1. (ب)–(ر) الكومة وفق الأكبر مباشرة بعد كل استدعاء لإجراء MAX-HEAPIFY في السطر 5، حيث تُعرَضُ قيمة i في ذلك الوقت. يبقى في الكومة العقد المظللة تظليلاً خفيفًا فقط. (ز) الصفيفة A المفروزة الناتجة.

5.6 الأرتال ذات الأولوية

يُعَدُّ الفرز بالكومة خوارزمية ممتازة، غير أننا سنعرض تنجيزًا جيدًا للفرز السريع في الفصل 7 يتفوّق عليها عمليًّا في العادة. ومع ذلك، فإن لبنية المعطيات "الكومة" نفسها استخدامات عديدة. نعرض، في هذا المقطع، إحدى أكثر تطبيقات الكومة شيوعًا، مثل: رتل أولويات فعال. وكما في الكومات، هناك نوعان من الأرتال ذات الأولويات: الأرتال ذات أولوية الأكبر max-priority queues، والأرتال ذات أولوية الأصغر min-priority queues. سنركز هنا على كيفية تنجيز الأرتال ذات أولوية الأكبر والتي تعتمد بدورها على

الكومات وفق الأكبر، يُطلَب إليك في التمرين 6.5-3 أن تكتب إجرائيات الأرتال ذات أولوية الأصغر.

الرتل ذو الأولوية priority queue هو بنية معطيات تسمح بتخزين مجموعة S من العناصر، يرفق مع كل منها قيمة تسمى ***مفتاحًا key***. يدعم ***الرتل ذو أولوية الأكبر max-priority queue*** العمليات التالية:

INSERT(S, x): تُدرِجُ العنصرَ x في المجموعة S، وهذا يكافئ العملية $S = S \cup \{x\}$.

MAXIMUM(S): تعيد العنصرَ ذا أكبر مفتاح من S.

EXTRACT-MAX(S): تَحذف العنصرَ ذا أكبر مفتاح من S وتعيده.

INCRESE-KEY(S, x, k): تَزيد قيمةَ مفتاح العنصر x إلى القيمة الجديدة k، التي يفترض أن تكون مساوية على الأقل قيمةَ مفتاح x الحالي.

يمكننا استخدام الأرتال ذات أولوية الأكبر، من بين تطبيقاتها الأخرى المتعددة، في جدولة المهام على حاسوب مشترك. يحتفظ الرتل ذو أولوية الأكبر بمسار المهام التي يجب إنجازها وأولوياتها الموافقة. عند انتهاء مهمة أو انقطاعها، يَختار المجدولُ scheduler المهمةَ ذات الأولوية العليا من بين المهام المُعلقة باستدعاء EXTRACT-MAX. يمكن للمجدول أن يضيف مهمةً جديدةً إلى الرتل في أي وقت باستدعاء INSERT.

وبالمقابل، يدعم الرتل ***ذو أولوية الأصغر min-priority queue*** العمليات INSERT و MINIMUM و EXTRACT-MIN و DECREASE-KEY. يمكن استخدام الرتل ذي أولوية الأصغر في محاكٍ مَقُودٍ بالأحداث event-driven simulator. فالعناصر في الرتل هي أحداث يجب محاكاتها، ويُرفَق مع كلٍّ منها زمنُ الحدوث الذي يُستخدم باعتباره مفتاحًا لها. ويجب محاكاة الأحداث وفق ترتيب زمن حدوثها، لأن محاكاة حدثٍ ما يمكن أن يسبِّب محاكاةَ أحداثٍ أخرى في المستقبل. يَستدعي برنامجُ المحاكاةِ إجراءَ EXTRACT-MIN في كلِّ مرحلةٍ لاختيار الحدث التالي الذي يجب محاكاته. كلّما تولدت أحداثٌ جديدة، يضيفها المجدولُ إلى الرتل ذي أولوية الأصغر باستدعاء INSERT. سنرى في الفصلين 23 و 24 استخدامات أخرى للأرتال ذات أولوية الأصغر تركز على عملية DECREASE-KEY.

من غير المستغرب أن يكون بإمكاننا استخدام الكومة لتنجيز رتلٍ ذي أولوية. ففي تطبيقٍ ما مثل جدولة المهام، أو محاكاةٍ مَقُودةٍ بالأحداث، توافق عناصرُ الرتل ذي الأولوية أغراضًا objects في ذلك التطبيق. وغالبًا ما نحتاج إلى تحديد غرض التطبيق الذي يوافق عنصرَ الرتل ذي الأولوية، والعكس بالعكس. لذلك فإننا غالبًا ما نحتاج – عند استخدام كومةٍ لتنجيز رتلٍ ذي أولوية – إلى تخزين ***مقبض handle*** يشير إلى غرض التطبيق الموافق في كل عنصر من الكومة. تعتمد بنية المقبض الفعلية (مثل مؤشر أو عدد صحيح) على التطبيق. وبالمثل، نحتاج إلى تخزين مقبض يشير إلى عنصر الكومة الموافق في كل غرضٍ من التطبيق. في هذه الحالة، يمكن أن يكون المقبضُ دليلَ صفيفة array index. عند التنجيز الفعلي، وبسبب تغيير عناصر الكومة لمواقعها ضمن الصفيفة خلال العمليات على الكومة، يجب – عند تغيُّر موقع عنصر في الكومة – تحديثُ (تعديل)

دليل الصفيفة في غرض التطبيق الموافق. ولما كانت تفاصيل النفاذ إلى أغراض التطبيقات تعتمد كثيرًا على التطبيق وعلى تنجيزه، فلن نتابع فيها هنا، بل سنؤكد فقط أنه لا بد من المحافظة - عمليًّا - على هذه المقابض بطريقةٍ صحيحة.

نناقش فيما يلي كيفية تنجيز عمليات الرتل ذي أولوية الأصغر. ينجّز الإجراءُ HEAP-MAXIMUM عملية MAXIMUM في زمن $\Theta(1)$.

```
HEAP-MAXIMUM(A)
1  return A[1]
```

وينجّز الإجراءُ HEAP-EXTRACT-MAX عمليةَ EXTRACT-MAX، وهو شبيهٌ بجسم حلقة **for** (الأسطر 3-5) من الإجراء HEAPSORT.

```
HEAP-EXTRACT-MAX(A)
1  if A.heap-size < 1
2      error "heap underflow"
3  max = A[1]
4  A[1] = A[A.heap-size]
5  A.heap-size = A.heap-size − 1
6  MAX-HEAPIFY(A, 1)
7  return max
```

إن زمن تنفيذ HEAP-EXTRACT-MAX هو $O(\lg n)$، لأنه يقوم بإنجاز عملٍ ثابتٍ فقط زيادةً على زمن MAX-HEAPIFY الذي هو $O(\lg n)$.

ينجّز الإجراءُ HEAP-INCREASE-KEY عمليةَ INCREACE-KEY. يُحدِّدُ دليلٌ i في الصفيفة عنصرَ الرتل ذي الأولوية الذي نرغب في زيادة قيمة مفتاحه. يعدِّل الإجراءُ أولاً قيمةَ مفتاح العنصر $A[i]$ إلى قيمته الجديدة. وحيث إن زيادةَ قيمةِ مفتاح $A[i]$ قد تَخرق خاصيةَ الكومة وفق الأكبر، فإن الإجراء HEAP-INCREASE-KEY يَعبُر (بطريقةٍ مشابهةٍ لحلقة الإدراج (في الأسطر 5-7) من INSERTION-SORT المذكورة في المقطع 1.2) مسارًا بسيطًا ابتداءً من هذه العقدة باتجاه الجذر لإيجاد المكان المناسب للمفتاح الذي جرت زيادة قيمته. ويقارِن HEAP-INCREASE-KEY - خلال عبوره هذا المسار - عنصرًا ما بأبيه بصورة تكرارية، ويبادِل بين مفتاحَيْهما، فيتابع إذا كان مفتاح العنصر أكبر، وينتهي إذا كان مفتاح العنصر أصغر، وذلك لأن خاصية الكومة وفق الأكبر أصبحت محققة الآن. (انظر التمرين 6.5-5 المتعلق بـ لامتغير الحلقة الدقيق.)

```
HEAP-INCREASE-KEY(A, i, key)
1  if key < A[i]
2      error "new key is smaller than current key"
3  A[i] = key
```

```
4  while i > 1 and A[PARENT(i)] < A[i]
5      exchange A[i] with A[PARENT(i)]
6      i = PARENT(i)
```

يُظهِر الشكل 5.6 مثالاً على كيفية عمل HEAP-INCREASE-KEY. إن زمن تنفيذ HEAP-INCREASE-KEY على كومة ذات n عنصرًا هو $O(\lg n)$، لأن طول المسار المرسوم من العقدة المُعدَّلة في السطر 3 وحتى الجذر هو $O(\lg n)$.

ينجِّز الإجراءُ MAX-HEAP-INSERT عمليةَ INSERT. إن دخْلَ هذا الإجراء هو مفتاحُ العنصر الجديد الذي سيُدرَج في الكومة وفق الأكبر A. يوسِّع الإجراءُ أولاً الكومةَ وفق الأكبر بإضافة ورقةٍ جديدةٍ مفتاحها $-\infty$ إلى الشجرة. بعد ذلك، يَستدعي HEAP-INCREASE-KEY لوضع القيمة الصحيحة في مفتاح العقدة الجديدة، وللحفاظ على خاصية الكومة وفق الأكبر.

```
MAX-HEAP-INSERT(A, key)
1  A.heap-size = A.heap-size + 1
2  A[A.heap-size] = -∞
3  HEAP-INCREASE-KEY(A, A.heap-size, key)
```

إن زمن تنفيذ MAX-HEAP-INSERT على كومةٍ ذات n عنصرًا هو $O(\lg n)$.

وبالجملة، فإن الكومةَ يمكنها أن تدعم أية عمليةٍ خاصةٍ بالأرتال ذات الأولوية على مجموعةٍ مؤلَّفةٍ من n عنصرًا في زمن $O(\lg n)$.

تمارين

1-5.6

اشرح كيفية عمل HEAP-EXTRACT-MAX على الكومة $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

2-5.6

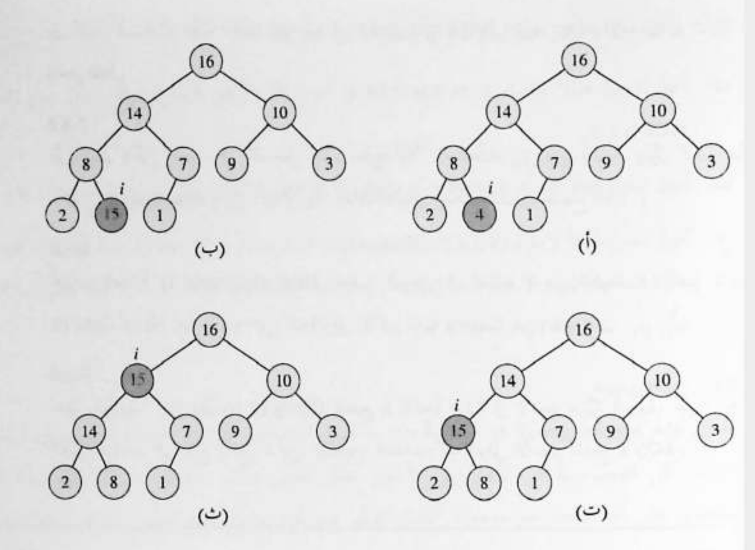
اشرح كيفية عمل MAX-HEAP-INSERT$(A, 10)$ على الكومة $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

3-5.6

اكتب شبه رماز للإجراءات التالية: HEAP-MINIMUM و HEAP-EXTRACT-MIN و HEAP-DECREASE-KEY و MIN-HEAP-INSERT، التي تنجز رتلاً ذا أولوية الأصغر باستخدام كومةٍ وفق الأصغر.

4-5.6

لماذا نتعب أنفسنا بوضع القيمة $-\infty$ في مفتاح العقدة الواجب إضافتها إلى السطر 2 من MAX-HEAP-INSERT عندما تكون العملية التالية التي سننفذها بعدها مباشرة هي زيادة مفتاح هذه العقدة إلى القيمة المطلوبة؟



الشكل 5.6 كيفية عمل HEAP-INCREASE-KEY. (أ) الكومة وفق الأكبر الموجودة في الشكل 4.6(أ) وقد ظُلِّلت فيها العقدة ذات الدليل i تظليلاً شديدًا. **(ب)** زِيدَتْ قيمةُ مفتاح هذه العقدة إلى 15. **(ت)** بعد تكرارٍ واحدٍ لحلقة **while** الموجودة في الأسطر 4-6، تبادلت العقدة وأبوها مفتاحَيْهما، وانتقل الدليل i صعودًا إلى الأب. **(ث)** الكومة وفق الأكبر بعد تكرارٍ آخرَ لحلقة **while**. لدينا الآن $A[\text{PARENT}(i)] \geq A[i]$، وأصبحت خاصية الكومة وفق الأكبر محقَّقة الآن، وينتهي الإجراء.

5-5.6

برهنْ صحةَ HEAP-INCREASE-KEY باستخدام لامتغير الحلقة التالي:

> عند بداية كلِّ تكرارٍ لحلقة **while** في الأسطر 4-6 لدينا $A[\text{PARENT}(i)] \geq A[\text{LEFT}(i)]$ و $A[\text{PARENT}(i)] \geq A[\text{RIGHT}(i)]$، إذا كانت هذه العقد موجودة وكانت الصفيفةُ الجزئية $A[1..A.heap\text{-}size]$ تُحقِّق خاصيةَ الكومة وفق الأكبر، باستثناء إمكان حصول خرق واحد: أن يكون $A[i]$ أكبر من $A[\text{PARENT}(i)]$.

يمكنك افتراض أن الصفيفة الجزئية $A[1..A.heap\text{-}size]$ تحقِّق خاصيةَ الكومة وفق الأكبر عند استدعاء HEAP-INCREASE-KEY.

6-5.6

تتطلَّب كلُّ عمليةِ تبديلٍ في السطر 5 من HEAP-INCREASE-KEY نموذجيًّا ثلاثة إسنادات assignments.

بيّن كيف نستخدم فكرة الحلقة الداخلية في INSERTION-SORT لتقليص هذه الإسنادات الثلاثة إلى إسنادٍ واحدٍ فقط.

7-5.6

بيِّن كيف يمكن تنجيز رتل "الداخل أولاً، خارج أولاً" باستخدام رتلٍ ذي أولوية. وبيِّن كيف يمكن تنجيز مكدسٍ stack باستخدام رتلٍ ذي أولوية. (عرَّفنا الأرتال والمكدسات في المقطع 1.10.)

8-5.6

تَحذف العمليةُ HEAP-DELETE(A, i) العنصرَ الموجود في العقدة i من الكومة A. أعطِ تنجيزًا لإجراء HEAP-DELETE زمنُ تنفيذه على كومةٍ وفق الأكبر ذات n عنصرًا هو $O(\lg n)$.

9-5.6

أعطِ خوارزميةً زمنُ تنفيذها $O(n \lg k)$ تدمج k لائحةً مرتبةً في لائحةٍ مرتبةٍ واحدة، حيث n هو العدد الكلي للعناصر في جميع لوائح الدخل. (*تلميح*: استخدم كومة وفق الأصغر لدمج k *لائحة*)

مسائل

1-6 بناء كومة باستخدام الإدراج

يمكننا بناء كومة بتكرار استدعاء MAX-HEAP-INSERT لإدراج العناصر في الكومة. لنأخذ النسخة المعدَّلة التالية من الإجراء BUILD-MAX-HEAP:

```
BUILD-MAX-HEAP'(A)
1  A.heap-size = 1
2  for i = 2 to A.length
3       MAX-HEAP-INSERT(A, A[i])
```

أ. هل ينشئ الإجراءان BUILD-MAX-HEAP و BUILD-MAX-HEAP' الكومةَ نفسَها دومًا عند تنفيذهما على صفيفة الدخل نفسها؟ برهن أن هذا صحيح، أو أعطِ مثالاً معاكسًا.

ب. بيِّن أن BUILD-MAX-HEAP' يتطلَّب في أسوأ الحالات زمنًا $\Theta(n \lg n)$ لبناء كومةٍ ذات n عنصرًا.

2-6 تحليل الكومات ذات d فرعًا

الكومة ذات d فرعًا d-ary heap تشبه الكومةَ الثنائية (باستثناء اختلافٍ وحيدٍ محتمل) وهو أن العقد المغايرة للأوراق لها d ابنًا بدلاً من ابنَيْن.

أ. كيف يمكنك تمثيل كومةٍ ذات d فرعًا باستخدام صفيفة؟

ب. ما هو ارتفاع كومة ذات d فرعًا و n عنصرًا بدلالة n و d؟

ت. أعطِ تنجيزًا فعالاً لإجراء EXTRACT-MAX في كومةٍ وفق الأكبر ذات d فرعًا. حلّلْ زمن تنفيذها بدلالة d و n.

ث. أعطِ تنجيزًا فعالاً لإجراء INSERT في كومةٍ وفق الأكبر ذات d فرعًا. حلّلْ زمن تنفيذها بدلالة d و n.

ج. أعطِ تنجيزًا فعالاً لإجراء INSREASE-KEY(A, i, k) الذي يعطي رسالة خطأ في حالة $k < A[i]$، وفي الحالة المعاكسة يُنفذ الإسناد $A[i] = k$، ثم يُعدِّل بنيةَ الكومة وفق الأكبر ذات d فرعًا بصورة ملائمة. حلِّلْ زمن تنفيذ هذا الإجراء بدلالة d و n.

3-6 جداول يونغ

جدول يونغ Young tableau هو مصفوفةٌ $m \times n$ بحيث أن عناصر كل سطر مرتبة من اليسار إلى اليمين، وعناصر كل عمود مرتبة من الأعلى إلى الأسفل. يمكن لبعض عناصر جدول يونغ أن تكون ∞، التي سنعاملها على أنها عناصر غير موجودة. لذلك، يمكن استخدام جدول يونغ لتخزين r عددًا منتهيًا حيث أن $r \leq mn$.

أ. ارسم جدول يونغ 4×4 يتضمن العناصر $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$.

ب. برهن أن جدول يونغ Y الذي بعداه $m \times n$ يكون خاليًا إذا كان $Y[1,1] = \infty$. وأن Y يكون ممتلئًا (أي يتضمن mn عنصرًا) إذا كان $Y[m, n] < \infty$.

ت. اكتب خوارزميةً لتنجيز EXTRACT-MIN على جدول يونغ $m \times n$ غير خالٍ، بحيث تُنفذ في زمن $O(m + n)$. يجب أن تَستخدم خوارزميتك مساقًا فرعيًا عوديًا يَحلُّ مسألةً $m \times n$ بحلّ مسألةٍ فرعيةٍ بُعداها $(m - 1) \times n$ أو $m \times (n - 1)$ عوديًا (*تلميح*: فكر بالإجراء MAX-HEAPIFY.) عَرِّف $T(p)$، حيث $p = m + n$، ليكون زمنَ التنفيذ الأعظم لإجراء EXTRACT-MIN على أي جدول يونغ $m \times n$. اكتب معادلةً تكرارية للزمن $T(p)$ تحقِّق الحدَّ الزمني $O(m + n)$، وحُلَّها.

ث. بيِّن كيفيةَ إدراج عنصرٍ جديد في جدول يونغ $m \times n$ غير ممتلئ في زمن $O(m + n)$.

ج. بيِّن، دون استخدام أية طريقة فرزٍ أخرى كمساق فرعي، كيفيةَ استخدام جدول يونغ بعداه $n \times n$ لفرز n^2 عددًا في زمن $O(n^3)$.

ح. اكتب خوارزميةً تنفَّذ في زمن $O(m + n)$ لتحديد كون عددٍ ما مخزَّنًا في جدول يونغ معطى، بعداه $m \times n$.

ملاحظات الفصل

ابتكر Williams [357] خوارزميةَ الفرز بالكومة، ووصَّف كذلك كيفية تنجيز رتلٍ ذي أولوية باستخدام كومة. واقترح Floyd في [106] الإجراء BUILD-MAX-HEAP.

نستخدم في الفصول 16 و 23 و 24 الكومات وفق الأصغر لتنجيز الأرتال ذات أولوية الأصغر. ونقدم في الفصل 19 تنجيزًا ذا حدود زمنية محسَّنة لعملياتٍ معيَّنة، وفي الفصل 20 افتراضٌ بأن المفاتيح مشتقة من مجموعة محدودة من أعداد صحيحة غير سالبة.

بيَّن Fredman و Willard في [115] كيفية تنجيز MINIMUM في زمن $O(1)$، وكيفية تنجيز INSERT و EXTRACT-MIN في زمن $O(\sqrt{\lg n})$، وذلك في حال كانت المعطيات أعدادًا صحيحة ذات b بتًّا، وكانت ذاكرة الحاسوب مؤلَّفة من كلماتٍ ذات b بتًّا قابلة للعنونة. وحسَّن Thorup في [337] الحدَّ $O(\sqrt{\lg n})$ ليصبح $O(\lg\lg n)$. يَستخدم هذا الحدُّ حجمَ تخزينٍ غير محدود بالقيمة n، ولكن يمكن تنجيزها بحجم خطي باستخدام تقطيع ذي عشوائية مضافة randomized hashing.

تحدث حالةٌ خاصةٌ هامةٌ من الأرتال ذات الأولوية عندما تكون متتالية العمليات EXTRACT-MIN ***مطردة monotone***، أي إن القيم التي تعيدها عمليات EXTRACT-MIN المتتالية تتزايد باطراد مع الزمن. تَظهر هذه الحالة في العديد من التطبيقات الهامة، مثل خوارمية Dijkstra لحساب أقصر مسار من منبع واحد، التي نناقشها في الفصل 24، وفي محاكاة الأحداث المتقطعة discrete-event simulation. من الضروري جدًّا في خوارزمية Dijkstra أن يجري، على وجه الخصوص، تنجيز عملية DECREASE-KEY بفاعلية. في حالة الاطراد وكون المعطيات أعدادًا صحيحة ضمن المجال $1, 2, \ldots, C$، وَصَّفَ Ahuja و Mehlhorn و Orlin و Tarjan في [8] كيفية تنجيز EXTRACT-MIN و INSERT في زمن مخمَّد $O(\lg C)$ (لمزيد من المعلومات عن التحليل المخمَّد انظر الفصل 17)، وكيفية تنجيز DECREASE-KEY في زمن $O(1)$، وذلك باستخدام بنية معطيات تسمى كومة الأسس radix heap. يمكن تحسين الحد $O(\lg C)$ ليصبح $O(\sqrt{\lg C})$ باستخدام كومات فيبوناتشي Fibonacci (انظر الفصل 19) إضافة إلى كومات الأسس. وأدخل Cherkassky و Goldberg و Silverstein في [66] تحسينًا إضافيًّا على هذا الحد ليصل إلى زمن متوقع $O(\lg^{1/3+\epsilon} C)$، وذلك بدمج بنية الدِّلاء المتعددة المستويات التي ابتكرها Denardo و Fox في [86] مع كومة Thorup المذكورة آنفًا. وأدخل Raman في [291] تحسينًا آخر على هذه النتائج لتصل إلى $O(\min(\lg^{1/4+\epsilon} C, \lg^{1/3+\epsilon} n))$ في حالة أي $\epsilon > 0$ محدد.

7 الفرز السريع

الفرز السريع خوارزميةٌ زمنُ تنفيذها في حالة صفيفة دخلٍ فيها n عددًا هو $\Theta(n^2)$ في أسوأ الحالات. وغالبًا ما يُعَدُّ الفرزُ السريع، على الرغم من بطء زمن تنفيذه في أسوأ الحالات، أفضلَ خيارٍ عمليٍّ للفرز، لأنه فعالٌ جدًّا في الحالة العامة: فزمنُ تنفيذه المتوقع هو $\Theta(n \lg n)$، والعواملُ الثابتة المخفية في تدوين $\Theta(n \lg n)$ صغيرةٌ جدًّا. ولهذه الخوارزمية أيضًا ميزةُ الفرز في المكان (انظر المقطع 1.2)، وهي تعمل جيدًا حتى في بيئات الذاكرة الافتراضية virtual memory.

يَصف المقطعُ 1.7 الخوارزميةَ ومساقًا فرعيًّا هامًّا يستخدمه الفرز السريع في عملية التجزئة partitioning. ونظرًا لتعقيد سلوك خوارزمية الفرز السريع، سنبدأ بمناقشةٍ بديهيةٍ لأدائها في المقطع 2.7 ونؤجِّل تحليلَها الدقيق إلى نهاية هذا الفصل. يَعرِض المقطع 3.7 نسخة من الفرز السريع تَستعمل عيِّناتٍ عشوائية. ولهذه الخوارزمية زمنُ تنفيذٍ متوقعٌ جيد، ولا يَستخلص دخلٌ خاصٌّ سلوكَها في أسوأ الحالات. يحلِّل المقطعُ 4.7 الخوارزميةَ ذات العشوائية المضافة randomized algorithm، حيث يبرهن أنها تُنَفَّذُ في زمن $\Theta(n^2)$ في أسوأ الحالات، وفي زمن متوقَّع $O(n \lg n)$ عند افتراض أن جميع عناصر الصفيفة متمايزة.

1.7 وصف الفرز السريع

يعتمد الفرزُ السريع، مثل الفرز بالمزج merge sort، مبدأ "فرِّق تسد" المذكور في المقطع 3.2-1. نعرض فيما يلي إجرائية "فرق تسد" ذات المراحل الثلاث لفرز صفيفة جزئية نموذجية $A[p\,..\,r]$:

فرِّق: جزِّئ (أعِدْ ترتيب) الصفيفة $A[p\,..\,r]$ إلى صفيفتين جزئيتين (يمكن أن تكونا خاليتين) $A[p\,..\,q-1]$ و $A[q+1\,..\,r]$ بحيث يكون كل عنصر من $A[p\,..\,q-1]$ أصغر من $A[q]$ أو يساويه، والذي هو بدوره، أصغر من أي عنصر من عناصر $A[q+1\,..\,r]$ أو يساويه. احسب الدليل q باعتباره جزءًا من إجراء التجزئة هذا.

سُدْ: افرز الصفيفتين الجزئيتين $A[p\,..\,q-1]$ و $A[q+1\,..\,r]$ باستدعاء عَوْدي للفرز السريع.

ادمج: لما كانت الصفيفتان الجزئيتان مفروزتين سلفًا، فلا داعيَ لدمجهما: فالصفيفة $A[p..r]$ كلُّها مفروزة الآن.

ينجِّز الإجراءُ التالي الفرزَ السريع.

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

لفرز كامل الصفيفة A، يكون الاستدعاء البدئي من الشكل QUICKSORT$(A, 1, A.length)$.

تجزئة الصفيفة

يُعَدُّ إجراءُ PARTITION، الذي يعيد ترتيب الصفيفة الجزئية $A[p..r]$ في المكان، أساسَ خوارزمية الفرز السريع.

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

يبين الشكل 1.7 كيف يعمل PARTITION على صفيفة من 8 عناصر. يختار PARTITION دائمًا عنصرًا $x = A[r]$ ليَكون *عنصرًا محوريًا* ***pivot*** يُجزِّأ الصفيفة الجزئية $A[p..r]$ بالنسبة إليه. يُجزِّئُ الإجراءُ، عند تشغيله، الصفيفةَ إلى أربع مناطق (يمكن أن تكون خالية). عند بداية كلِّ تكرارٍ من حلقة **for** في السطور 3-6، تحقِّق المناطقُ خواصَّ محدَّدةً مبيَّنةً في الشكل 2.7. نعتبر هذه الخواص لامتغير الحلقة loop invariant:

لدينا في بداية كل تكرار من الحلقة في السطور 3-6، ولكل دليلٍ k من الصفيفة:

1. إذا كان $p \le k \le i$ فإن $A[k] \le x$.
2. إذا كان $i + 1 \le k \le j - 1$ فإن $A[k] > x$.
3. إذا كان $k = r$ فإن $A[k] = x$.

(أ) i p,j … r: | 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4 |

(ب) p,i j … r: | 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4 |

(ت) p,i j … r: | 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4 |

(ث) p,i j … r: | 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4 |

(ج) p i j r: | 2 | 1 | 7 | 8 | 3 | 5 | 6 | 4 |

(ح) p i j r: | 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4 |

(خ) p i j r: | 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4 |

(د) p i r: | 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4 |

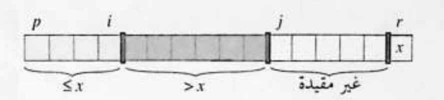
(ذ) p i r: | 2 | 1 | 3 | 4 | 7 | 5 | 6 | 8 |

الشكل 1.7 عملية تطبيق التجزئة PARTITION على صفيفة عيّنة. يصبح عنصر الصفيفة $A[r]$ العنصر المحوري x. تقع جميع عناصر الصفيفة الخفيفة التظليل في الجزء الأول، وقيمها لا تتجاوز x. وتقع العناصر الشديدة التظليل في الجزء الثاني، وقيمها أكبر من x. لم توضع العناصر غير المظللة بعدُ في أول جزأين، والعنصر الأبيض الأخير هو المحور x. (أ) الصفيفة البدائية ومواضع المتحولات. لم يوضع أي من العناصر في أول جزأين. **(ب)** جرى تبديل موقع القيمة 2 "مع نفسه"، ووضعت في الجزء الخاص بالقيم الصغرى. **(ت)**–**(ث)** أضيفت القيمتان 8 و 7 إلى الجزء الخاص بالقيم الكبرى. **(ج)** جرى تبديل موقعي القيمتين 1 و 8، وكَبُرَ جزء القيم الصغرى. **(ح)** جرى تبديل موقعي 3 و 7، وكَبُرَ جزء القيم الصغرى. **(خ)**–**(د)** كَبُرَ الجزء الأكبر ليتضمن 5 و 6 وتنتهي الحلقة. **(ذ)** في السطرين 7-8، جرى تبديل موقع المحور بحيث يقع بين الجزأين.

لا تُغطَّى الأدلةُ الواقعة بين j و $r-1$ بأيٍّ من الحالات الثلاث، وليس لقيم هذه العناصر أية علاقة مع المحور x.

ينبغي أن نبيّن أن لامتغير الحلقة هذا صحيحٌ true قبل التكرار الأول، وأن كل تكرارٍ لهذه الحلقة يحافظ على هذا اللامتغير، الذي يقدم خواصَّ مفيدةً لبيان الصحة correctness عند انتهاء الحلقة.

الاستبداء: لدينا قبل التكرار الأول للحلقة، $i=p-1$ و $j=p$. وحيث إنه لا توجد قيمٌ بين p و i، ولا قيمٌ بين $i+1$ و $j-1$، فإن أول شرطَيْن خاصَّيْن بلامتغير الحلقة محقَّقان بديهيًّا. يُحقق الإسنادُ في السطر 1 الشرطَ الثالث.



الشكل 2.7 المناطق الأربع التي يحافظ عليها إجراء PARTITION ضمن الصفيفة الجزئية $A[p\,..\,r]$. جميع القيم في $A[p\,..\,i]$ أصغر من x أو تساويها، وجميع القيم في $A[i+1\,..\,j-1]$ أكبر من x، و $A[r] = x$. يمكن أن تأخذ العناصر في الصفيفة الجزئية $A[j\,..\,r-1]$ أية قيمة.

المحافظة: يبين الشكل 3.7، أننا نعتبر حالتين اعتمادًا على خرج الاختبار في السطر 4. يبين الشكل 3.7(أ) ماذا يحدث عندما يكون $A[j] > x$؛ الفعل الوحيد في الحلقة هو زيادة قيمة j. بعد زيادة j، يبقى الشرط 2 محققًا في حالة $A[j-1]$، وتبقى جميع العناصر الأخرى دون تعديل. يبين الشكل 7-3(ب) ماذا يحدث عندما تكون $A[j] \le x$؛ حيث تزيد الحلقة قيمة i، وتبدِّل موقعَي $A[i]$ و $A[j]$، ثم تزيد قيمة j. لدينا الآن، بسبب تبديل المواقع، $A[i] \le x$، ويكون الشرط 1 محقَّقًا. وبالمثل، يكون لدينا أيضًا $A[j-1] > x$، لأن العنصرَ الذي وَضَعَهُ لامتغيرُ الحلقة في $A[j-1]$ أكبرُ من x.

الانتهاء: عند الانتهاء يكون لدينا $j = r$. لذا، يوجد كلُّ عنصرٍ في الصفيفة في إحدى المجموعات الثلاث الموصوفة باللامتغير، ونكون قد جزأنا القيم في الصفيفة إلى ثلاث مجموعات: أقل أو تساوي x، وأكبر من x، ومجموعة فيها عنصر وحيد هو x.

يبدِّل السطران الأخيران من PARTITION في النهاية العنصرَ المحوري بالعنصر الموجود في أقصى يسار المجموعة التي قيمها أكبر من x، وبذلك تنقل المحور إلى مكانه الصحيح في الصفيفة المجزأة، ثم تعيد الدليل الجديد للمحور. يحقِّق خرجُ PARTITION الآن المواصفات المحددة لمرحلة "فرِّق". في الحقيقة، يحقق الخرج شرطًا أقوى بقليل: بعد السطر 2 من QUICKSORT، يكون $A[q]$ أصغر تمامًا من أي عنصر في $A[q+1\,..\,r]$.

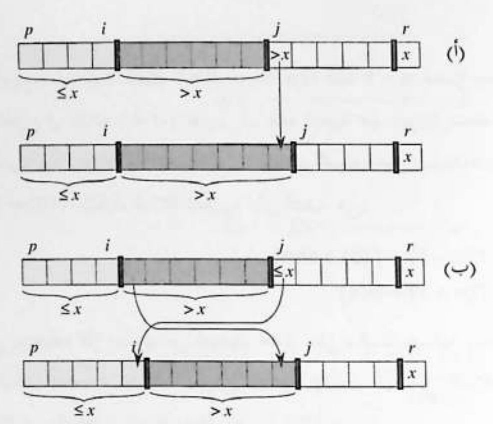
إن زمن تنفيذ PARTITION على الصفيفة الجزئية $A[p\,..\,r]$ هو $\Theta(n)$، حيث $n = r - p + 1$ (انظر التمرين 7. 1-3).

تمارين

1-1.7

وضِّح، باستخدام الشكل 1.7 نموذجًا، كيفية تطبيق التجزئة PARTITION على الصفيفة

$A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$



الشكل 3.7 حالتا تكرارٍ واحدٍ للإجراء PARTITION. (أ) إذا كان $A[j] > x$، فالفعل الوحيد هو زيادة قيمة j، الذي يحافظ على لامتغير الحلقة. (ب) إذا كان $A[j] \leq x$، تُزاد قيمة الدليل i، ثم يجري تبديل مكانيَ العنصرين $A[i]$ و $A[j]$، ثم تُزاد قيمة j. جرت المحافظة على لامتغير الحلقة ثانيةً.

2-1.7

ما قيمة q التي يعيدها إجراء PARTITION عندما تكون قيم جميع عناصر الصفيفة $A[p\,..r]$ متساوية؟ عدِّل PARTITION بحيث يكون $q = \lfloor (p+r)/2 \rfloor$ عندما تكون قيم جميع العناصر في الصفيفة $A[p\,..r]$ متساوية.

3-1.7

أعط برهانًا مختصرًا على أن زمن تنفيذ PARTITION على صفيفة جزئية طولها n هو $\Theta(n)$.

4-1.7

كيف يمكنك تعديل QUICKSORT لإجراء الفرز وفق الترتيب المتناقص؟

2.7 أداء الفرز السريع

يعتمد زمن تنفيذ الفرز السريع على كون التجزئة متوازنة أو غير متوازنة، وهذا بدوره يعتمد على العناصر المستخدمة في التجزئة. فإذا كانت التجزئة متوازنة، تكون سرعة تنفيذ الخوارزمية مقاربة لسرعة البحث بالمزج. أما إذا كانت غير متوازنة، فيمكن أن تنفَّذ الخوارزمية ببطءٍ مقاربٍ للفرز بالإدراج. سنبحث في هذا المقطع، بصورة غير رسمية (مفصلة)، في أداء الفرز السريع بافتراض أن التجزئة المتوازنة مقارنة بأدائه عند التجزئة غير المتوازنة.

التجزئة في أسوأ الحالات

تحدث أسوأ حالات الفرز السريع عندما يولِّد مساقُ التجزئةِ مسألةً جزئية فيها $n-1$ عنصرًا ومسألةً ليس فيها أي عنصر. (نُثْبت هذا الطرح في المقطع 7.4-1.) لنفترض أن هذه التجزئة غير المتوازنة ستحدث في كل استدعاء عودي. تستغرق التجزئة زمنًا $\Theta(n)$. ولما كان تطبيق الاستدعاء العودي على صفيفة طولها 0 يخرج من الإجراء فقط، فإن $T(0) = \Theta(1)$، وتكون العلاقة التكرارية لزمن التنفيذ هي:

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \ . \end{aligned}$$

بديهيًّا، إذا جمعنا الأزمنة التي يستغرقها كلُّ مستوى من العودية، نحصل على سلسلة حسابية (المعادلة أ.2)؛ قيمتها $\Theta(n^2)$. بالفعل، يمكن استخدام طريقة التعويض مباشرة لإثبات أن حل العلاقة التكرارية $T(n) = T(n-1) + \Theta(n)$ هو $T(n) = \Theta(n^2)$. (انظر التمرين 7.2-1.)

أي إنه، إذا كانت التجزئة غير متوازنة كليًّا في كل مستوى عودي من الخوارزمية، فإن زمن التنفيذ هو $\Theta(n^2)$. ولذلك يكون زمن تنفيذ الفرز السريع في أسوأ الحالات ليس أفضل من الفرز بالإدراج. إضافة إلى ذلك، يكون زمن التنفيذ $\Theta(n^2)$ عندما تكون صفيفةُ الدخل مفروزةً كليًّا سلفًا – وهي حالةٌ شائعةٌ زمنُ تنفيذ الفرز بالإدراج فيها هو $O(n)$.

التجزئة في أحسن الحالات

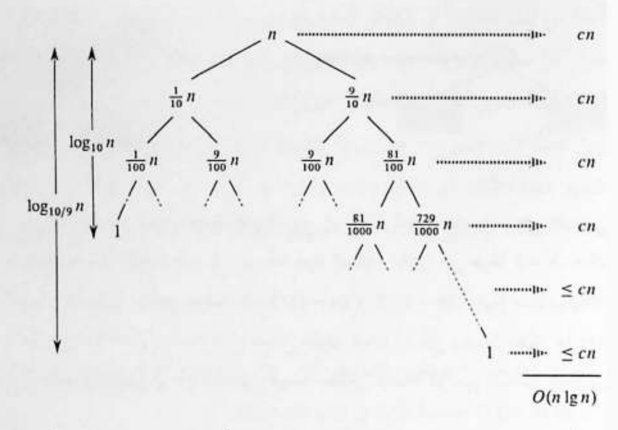
يولِّد PARTITION، في الحالة التي يجري فيها التفريق إلى جزأين أكثر ما يكونان متوازنين، مسألتين فرعيتين، لا يتجاوز طول كل منهما $n/2$، حيث إن طول إحداها $\lfloor n/2 \rfloor$ وطول الثانية $\lceil n/2 \rceil - 1$. في هذه الحالة، تُنَفَّذُ خوارزمية الفرز السريع أسرع بكثير. وتكون المعادلة التكرارية لزمن التنفيذ عندها:

$$T(n) = 2T(n/2) + \Theta(n) \ ,$$

حيث نتساهل في عدم الدقة الناتج من إهمال دالة الأرضية floor ودالة السقف ceiling ومن طرح القيمة 1. حَلُّ هذه المعادلة التكرارية حسب الحالة الثانية من النظرية العامة (النظرية 4.1) هو $T(n) = \Theta(n \lg n)$. وهكذا، فإن موازنة طرفي التجزئة بصورة متساوية في كل مستوى من العودية تعطينا خوارزميةً أسرعَ تقاربيًّا.

التجزئة المتوازنة

إن زمن التنفيذ في الحالة الوسطى للفرز السريع أقرب كثيرًا إلى الحالة المثلى منه إلى أسوأ الحالات، وهو ما ستبيِّنه التحليلات في المقطع 7.4. إن مفتاح فهم السبب هو في فهم كيف يؤثِّر توازن التجزئة على المعادلة التكرارية التي توصّف زمن التنفيذ.



الشكل 4.7 شجرة عودية لخوارزمية QUICKSORT تولِّد فيها PARTITION تفريقاً بنسبة 9 إلى 1 دومًا، وهذا يولِّد زمن تنفيذ $O(n \lg n)$. تبيِّن العقد حجوم المسائل الجزئية، وقد جرى وضع كلفة كل مستوى إلى اليمين. تتضمن كلفة كل مستوى الثابت c الضمني في الحد $\Theta(n)$.

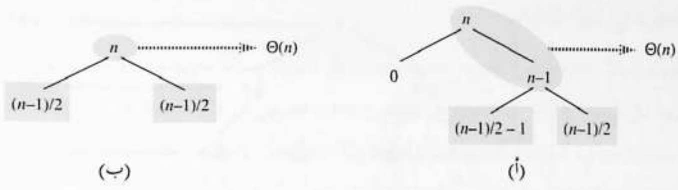
لنفترض، مثلاً، أن خوارزمية التجزئة تولِّد دومًا جزأين نسبة أحدهما إلى الآخر 9 إلى 1، الشيء الذي يبدو للوهلة الأولى غير متوازن إلى درجة كبيرة. نحصل في هذه الحالة على المعادلة التكرارية التالية لزمن تنفيذ الفرز السريع:

$$T(n) = T(9n/10) + T(n/10) + cn \ ,$$

حيث أضفنا الثابت c صراحة بعد أن كان مخفيًّا في الحد $\Theta(n)$. يبين الشكل 4.7 شجرة العودية لهذه المعادلة التكرارية. لاحظ أن كلفة كل مستوى من الشجرة تساوي cn، إلى أن تحقِّق العودية شرطًا حديًّا عند العمق $\log_{10} n = \Theta(\lg n)$، تصبح بعدها كلفة كل مستوى مساوية cn على الأكثر. تنتهي العودية عند العمق $\log_{10/9} n = \Theta(\lg n)$. وبذلك تكون الكلفة الكلية للفرز السريع $O(n \lg n)$. ومن ثمّ، ومع تفريق نسبته 9 إلى 1 في كل مستوى من العودية، وهو ما يبدو بديهيًّا غير متوازن إلى حدٍّ بعيد، فإن خوارزمية الفرز السريع تُنفَّذ في زمن $O(n \lg n)$ - أي في زمن مقارب للحالة التي تجري فيها عملية التفريق في الوسط. فعليًّا، حتى عملية التفريق الذي نسبته 99 إلى 1 تنتج زمن تنفيذ $O(n \lg n)$. في الحقيقة، أي تفريق نسبته *ثابتة* ينتج شجرة عودية عمقها $\Theta(\lg n)$، حيث كلفة كل مستوى $O(n)$. بالنتيجة، يكون زمن التنفيذ $O(n \lg n)$ في حال كان التفريق يجري بنسبة ثابتة.

حدس بشأن الحالة الوسطى

لتكوين فكرة واضحة عن السلوك العشوائي للفرز السريع، يجب أن نضع فرضية عن مدى التواتر الذي نتوقع



الشكل 5.7 (أ) مستويان من شجرة العودية للفرز السريع. إن تكلفة التجزئة عند الجذر هي n، وهي تولِّد تفريقاً "غير جيد": صفيفتَيْن جزئيتين طولهما 0 و $n-1$. أما تجزئة الصفيفة الجزئية التي طولها $n-1$، فتكلف $n-1$، وتولد تفريقاً "جيداً": صفيفتَيْن جزئيتين طولهما $(n-1)/2-1$ و $(n-1)/2$. (ب) مستوى واحد من شجرة العودية متوازن جدًّا. في كلا الجزأين، كلفة تجزئة المسائل الجزئية الممثلة بشكلٍ بيضويٍّ مظلَّلٍ هي $\Theta(n)$. ولكن المسألتين الجزئيتين الواجب حلهما في (أ) والممثلتين بمربعين مظلَّلين ليستا أكبر من المسألتين الجزئيتين اللتين يلزم حلهما في (ب).

أن نصادف فيه المدخلات المختلفة. يعتمد سلوك الفرز السريع على الترتيب النسبي للقيم في عناصر الصفيفة المعطاة باعتبارها دخلاً، وليس على القيم الخاصة في الصفيفة. سنفترض، كما في تحليلنا الاحتمالي لمسألة التوظيف في المقطع 2.5، أن كل تباديل أعداد الدخل متساوية الاحتمال.

عندما ننفذ الفرز السريع على صفيفةِ دخلٍ قيمُها عشوائية، فمن غير المحتمل أن تجري التجزئة بالطريقة نفسها في كل مستوى، كما افترض تحليلنا غير الرسمي. من المنطقي أن نتوقع أن بعض حالات التفريق ستكون جيدة التوازن، وبعضها ستكون متوازنة قليلاً. على سبيل المثال، يُطلَب إليك في التمرين 2.7-6 إثبات أنه في 80 بالمئة من المرات تقريبًا يولِّد PARTITION تفريقاً أكثر توازنًا من 9 إلى 1، وفي 20 بالمئة من المرات تقريبًا يولِّد تفريقاً أقل توازنًا من 9 إلى 1.

تولِّد PARTITION، في الحالة الوسطى، مزيجًا من التفاريق "الجيدة" و"غير الجيدة". وفي الشجرة العودية الخاصة بتنفيذ PARTITION في الحالة الوسطى، تتوزَّع التفاريق الجيدة وغير الجيدة عشوائيًّا في جميع أرجاء الشجرة. ولكن، لنفترض للتبسيط أن التفاريق الجيدة وغير الجيدة تتبادلان المستويات في الشجرة، وأن التفاريق الجيدة هي من أحسن الحالات، و التفاريق غير الجيدة هي من أسوأ الحالات. يبيِّن الشكل 5.7(أ) التفاريق على مستويين متتاليين من شجرة العودية. إن كلفة التجزئة عند جذر الشجرة هي n، وطول الصفيفتين الجزئيتين هو $n-1$ و 0: وفق أسوأ الحالات. وفي المستوى التالي، تُجزَّأ الصفيفة الجزئية التي طولها $n-1$ وفق أحسن الحالات إلى صفيفتين جزئيتين طولهما $(n-1)/2-1$ و $(n-1)/2$. لنفترض أن كلفة الشرط الحدِّي هي 1 للصفيفة الجزئية التي طولها 0.

يولِّد تركيبُ التفريق غير الجيد مع التفريق الجيد ثلاث صفيفات جزئية أطوالها: 0 و $(n-1)/2-1$ و $(n-1)/2$ بكلفةِ تجزئةٍ مركَّبة تساوي $\Theta(n)+\Theta(n-1)=\Theta(n)$. إن هذه الحالة ليست بالتأكيد

أسوأ من تلك المذكورة في الشكل 7-5(ب)، أي حالة مستوى واحد من التجزئة التي تولّد صفيفتين جزئيتين طول كلٍّ منهما $(n-1)/2$، وكلفة $\Theta(n)$. ومع ذلك، فإن هذه الحالة الأخيرة متوازنة! ومن البديهي أن تمتص كلفةُ التفريق الجيد $\Theta(n)$ كلفةَ التفريق غير الجيد $\Theta(n-1)$، ويكون التفريق الناتج جيداً. وهكذا، فإن زمنَ تنفيذ الفرز السريع – عند تبديل المستويات بين تفريق جيد وغير جيد – يماثِل زمنَ التنفيذ عند إجراء التفريق الجيد فقط: أي $O(n \lg n)$، ولكن مع ثابتٍ أكبر قليلاً مخفي ضمن تدوين-O. سنجري تحليلاً مفصلاً للحالة الوسطى للنسخة ذي العشوائية المضافة للفرز السريع في المقطع 7.4-2.

تمارين

1-2.7

استخدم طريقة التعويض لإثبات أن حل المعادلة التكرارية $T(n) = T(n-1) + \Theta(n)$ هو $T(n) = \Theta(n^2)$، حسبما ذكرنا في بداية المقطع 2.7.

2-2.7

ما هو زمن تنفيذ QUICKSORT عندما تكون لكل عناصر الصفيفة A القيمة نفسها؟

3-2.7

برهن أن زمن تنفيذ QUICKSORT هو $\Theta(n^2)$ عندما تتضمن الصفيفة A عناصر متمايزة ومرتَّبة وفق الترتيب التُّنَزولي.

4-2.7

تسجّل المصارف عادة التداولات على حسابٍ ما وفق ترتيب زمن التداول، ولكن يرغب العديد من الناس أن يتلقوا بياناتهم المصرفية بحيث تكون الشيكات مرتبة وفق رقم الشيك. يحرّر الناس عادة الشيكات وفق ترتيب أرقام الشيكات، ويَصرف الباعة هذه الشيكات عادة بعد فترة معقولة. إن مسألة تحويل الترتيب وفق زمن التداول إلى ترتيب وفق رقم الشيك هي مسألة فرز دخلٍ مفروز تقريبًا. برهن أن إجراء INSERTION-SORT قد يتفوّق على إجراء QUICKSORT في هذه المسألة.

5-2.7

نفترض أن نسبة التفاريق على كل مستوى من الفرز السريع هي $1-\alpha$ إلى α، حيث $0 < \alpha \le 1/2$ ثابت. بيِّن أن العمق الأصغري لورقةٍ ما في شجرة العودية هو تقريبًا $-\lg n / \lg \alpha$ وأن العمق الأعظمي هو $-\lg n / \lg(1-\alpha)$ تقريبًا. (لا تهتم بشأن تدوير العدد الصحيح.)

6-2.7 ⋆

برهن أن احتمال أن يولّد PARTITION تفريقاً أكثرَ توازنًا من $1-\alpha$ إلى α على صفيفة دخلٍ عشوائية هو $1-2\alpha$ تقريبًا، وذلك مهما يكن الثابت $0 < \alpha \le 1/2$.

3.7 نسخةٌ للفرز السريع ذو عشوائيةٍ مضافة

عند سبر سلوك الحالة الوسطى للفرز السريع، افترضنا أن جميع تباديل أعداد الدخل ذات احتمال متساو. ولكننا لا نستطيع أن نتوقع تحقُّق ذلك دائمًا في مسألة هندسية (انظر التمرين 7.2-4). يمكننا في بعض الأحيان - كما رأينا في المقطع 5.3 - إضافةُ عشوائيةٍ إلى خوارزمية بهدف الحصول على أداء جيد في الحالة الوسطى على جميع المدخلات. هذا وينظر كثيرون إلى نسخة الفرز السريع ذي العشوائية الناتجة على أنها خوارزمية الفرز الأنسب في حالة مدخلات كبيرة كفاية.

قمنا في المقطع 5.3، بإضافة عشوائيةٍ إلى خوارزميتنا، وذلك بتبديل عناصر الدخل صراحة. وكان بإمكاننا إجراء ذلك للفرز السريع أيضًا، ولكن تقنيةَ إضافة عشوائيةٍ مختلفة، تسمى ***اختيار عينات عشوائي random sampling***، تعطي تحليلاً أبسط. فبدلاً من استخدام $A[r]$ محورًا على الدوام، نستخدم عنصرًا جرى اختياره عشوائيًّا من الصفيفة الجزئية $A[p\,..\,r]$. نُجري ذلك بتبديل موقع العنصر $A[r]$ بعنصر جرى اختياره عشوائيًّا من $A[p\,..\,r]$. يَضمن أخذُ عيناتٍ عشوائية من المجال $p,\ldots,r$، أن يكون العنصر المحوري $x = A[r]$ أيًّا من عناصر الصفيفة الجزئية التي عددها $r - p + 1$، وذلك باحتمال متساوٍ. ولما كان اختيار العنصر المحوري قد جرى عشوائيًّا، فنحن نتوقع أن يكون تفريق صفيفة الدخل متوازناً وسطيًّا إلى حدٍّ معقول.

إن التعديلات التي تُجرى على PARTITION و QUICKSORT طفيفة؛ ففي إجراء التجزئة الجديد، ما علينا سوى تنجيز تبديل المواقع قبل إجراء التجزئة الحالي:

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

يستدعى الفرز السريع الجديد RANDOMIZED-PARTITION بدلاً من PARTITON:

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2       q = RANDOMIZED-PARTITION(A, p, r)
3       RANDOMIZED-QUICKSORT(A, p, q − 1)
4       RANDOMIZED-QUICKSORT(A, q + 1, r)
```

سنحلِّل هذه الخوارزمية في المقطع التالي.

تمارين

7.3-1

لماذا نقوم بتحليل زمن التنفيذ المتوقع للخوارزمية ذات العشوائية المضافة وليس زمن التنفيذ في أسوأ الحالات؟

2-3.7
خلال تنفيذ RANDOMIZED-QUICKSORT، كم مرة يُستدعى مولِّد الأعداد العشوائية RANDOM في أسوأ الحالات؟ وكم مرة في أفضل الحالات؟ أعطِ إجابتك باستخدام تدوين-Θ.

4.7 تحليل الفرز السريع

قدم المقطع 2.7 بعض الأمور البديهية المتعلقة بسلوك الفرز السريع في أسوأ الحالات، ولماذا نتوقع أن تنفَّذ بسرعة. وفي هذا المقطع، نحلِّل سلوكَ الفرز السريع تحليلاً دقيقًا. نبدأ بالتحليل في أسوأ الحالات، الذي ينطبق على QUICKSORT أو RANDOMIZED-QUICKSORT، ونختم بتحليل زمن التنفيذ المتوقع لإجراء RANDOMIZED-QUICKSORT.

1.4.7 التحليل في أسوأ الحالات

رأينا في المقطع 2.7 أن إجراء التفريق في أسوأ الحالات في كل مستوى من مستويات العودية في الفرز السريع يولِّد زمن تنفيذ $\Theta(n^2)$، وهو – بديهيًا – زمن تنفيذ الخوارزمية في أسوأ الحالات. نبرهن فيما يلي هذه الفرضية المؤكَّدة:

يمكننا، باستعمال طريقة التعويض (انظر المقطع 3.4)، أن نبيِّن أن زمن تنفيذ الفرز السريع هو $\Theta(n^2)$. ليكن $T(n)$ زمن أسوأ الحالات لإجراء QUICKSORT على دخل طوله n. لدينا المعادلة التكرارية:

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n) \ , \qquad (1.7)$$

حيث تقع قيمة المُوسِط q بين 0 و $n-1$، وذلك لأن إجراء PARTITION يولِّد مسألتين جزئيتين طولهما الكلي $n-1$. نتوقع أن يكون $T(n) \le cn^2$، حيث c ثابت ما. وبتعويض هذا التوقع في المعادلة (1.7)، نحصل على:

$$\begin{aligned} T(n) &\le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n) \ . \end{aligned}$$

يبلغ التعبير $q^2 + (n-q-1)^2$ قيمته العظمى على مجال المُوسِط $0 \le q \le n-1$ عند نقطتَيْه الطرفيَّتَيْن. ولإثبات هذا الإدعاء، نلاحظ أن المشتق الثاني للتعبير بالنسبة إلى q موجب (انظر التمرين 3-4.7)، وهذه الملاحظة تعطينا الحدَّ $\max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) \le (n-1)^2 = n^2 - 2n + 1$ وبمتابعة عملية وضع حد لـ $T(n)$، نحصل على:

$$\begin{aligned} T(n) &\le cn^2 - c(2n-1) + \Theta(n) \\ &\le cn^2 \ , \end{aligned}$$

وذلك لأن بإمكاننا اختيار الثابت c كبيرًا كفاية بحيث يطغى الحد $c(2n-1)$ على الحد $\Theta(n)$. وبذلك يكون $T(n) = O(n^2)$. وقد رأينا في المقطع 2.7 حالةً خاصة يستغرق فيها الفرز السريع زمنًا $\Omega(n^2)$، وذلك عندما تكون التجزئة غير متوازنة. وبالمقابل، يُطلب إليك في التمرين 7.4-1 برهان أن للمعادلة التكرارية (7.1) حلاً يحقِّق $T(n) = \Omega(n^2)$. وبذلك، يكون زمن التنفيذ للفرز السريع في أسوأ الحالات هو $\Theta(n^2)$.

7.4.2 زمن التنفيذ المتوقع

رأينا فيما سبق لمَ يكون الزمن المتوقع لتنفيذ خوارزمية RANDOMIZED-QUIKSORT في أسوأ الحالات هو $O(n \lg n)$: إذا كان التفريق الناجم عن RANDOMIZED-PARTITION في كل مستوى من العودية، يضع أية نسبة ثابتة من العناصر في جهة واحدة من التجزئة، يكون لشجرة العودية العمق $\Theta(\lg n)$، كما يجري تنفيذ أعمال من رتبة $O(n)$ في كل مستوى. حتى لو أضفنا بين هذه المستويات مستويات جديدة التفريق فيها قليل التوازن إلى حد بعيد، فسيبقى الزمن الكلي $O(n \lg n)$. يمكننا تحليل زمن التنفيذ المتوقع للإجراء RANDOMIZED-QUICKSORT بدقة إذا عرفنا كيف يعمل إجراء التجزئة أولاً، ثم استخدمنا هذه المعرفة لاستنتاج حدٍّ $O(n \lg n)$ لزمن التنفيذ المتوقع. يولِّد هذا الحد الأعلى لزمن التنفيذ المتوقع، إضافةً إلى الحد في أفضل الحالات $\Theta(n \lg n)$، الذي ذكرناه في المقطع 7.2، زمن تنفيذ متوقع $\Theta(n \lg n)$. نفترض في أثناء ذلك أن قيم العناصر المفروزة متمايزة.

زمن التنفيذ ومقارنات

الاختلاف الوحيد بين إجراءَي QUICKSORT و RANDOMIZED-QUICKSORT هو في كيفية اختيار العناصر المحورية؛ وهما متماثلان فيما عدا ذلك. لذا، يمكننا صياغة تحليلنا لإجراء RANDOMIZED-QUICKSORT بمناقشة إجراءَي QUICKSORT و PARTITION، ولكن بافتراض أن اختيار العناصر المحورية يجري عشوائيًّا من الصفيفة الجزئية الممررة إلى RANDOMIZED-QUICKSORT.

يهيمن الزمن الذي يقضيه إجراء PARTITION على زمن تنفيذ QUICKSORT. ففي كلّ استدعاءٍ لإجراء PARTITION، يختار عنصرًا محوريًّا، ولا يُضمَّن هذا العنصر في أيِّ استدعاءٍ عودي مستقبلي لإجراءَي QUICKSORT و PARTITION. لذلك، يمكن أن يكون هناك n استدعاءً على الأكثر لإجراء PARTITION خلال التنفيذ الكامل لخوارزمية الفرز السريع. يَستغرق استدعاءٌ واحد لإجراء PARTITION زمنًا $O(1)$ إضافةً إلى زمنٍ متناسبٍ طردًا مع عدد تكرارات حلقة **for** الموجودة في الأسطر 3-6. يُجرِي كلُّ تكرارٍ لحلقة **for** هذه مقارنةً في السطر 4، يقارِن فيها المحورَ بعنصرٍ آخر من الصفيفة A. لهذا، إذا كان بإمكاننا عد العدد الكلي لمرات تنفيذ السطر 4، يمكننا وضع حدٍّ للزمن الكلي الذي تستغرقه حلقة **for** خلال التنفيذ الكامل لإجراء QUICKSORT.

توطئة 1.7

ليكن X عدد المقارنات التي أُجريت في السطر 4 من PARTITION خلال التنفيذ الكامل لإجراء QUICKSORT على صفيفة ذات n عنصرًا. عندها، يكون زمن تنفيذ QUICKSORT هو $O(n + X)$.

البرهان نستنتج من المناقشة السابقة أن الخوارزمية تستدعي PARTITION n مرةً على الأكثر، يُنجَز في كلٍّ منها مقدارٌ ثابتٌ من الأعمال، ثم تنفَّذ حلقة **for** عددًا من المرات. ويُنفِّذُ كلُّ تكرار لحلقة **for** السطرَ 4. ■

غايتنا إذن حساب X، وهو عدد المقارنات الكلية المنفَّذة في جميع استدعاءات PARTITION. ولن نحاول تحليل كم هو عدد المقارنات التي أُجريت عند كلِّ استدعاء للإجراء PARTITION. بل، سنستنتج بدلاً من ذلك، حدًّا شموليًا لعدد المقارنات الكلي. ولِفعل ذلك، علينا أن نَعرف متى تقارِنُ الخوارزميةُ بين عنصرين من الصفيفة، ومتى لا تقارِن. نسمِّي، لسهولة التحليل، عناصرَ الصفيفة A بـ $z_1, z_2, \ldots, z_n$، حيث z_i هو العنصر ذو الترتيب i من حيث الصغر. ونُعرِّف أيضًا المجموعةَ $Z_{ij} = [z_i, z_{i+1}, \ldots, z_j]$ بأنها مجموعة العناصر z_i و z_j وما بينهما.

والسؤال هو: متى تقارِنُ الخوارزميةُ بين z_i و z_j؟ للإجابة عن هذا السؤال، نلاحظ أولاً أنه تجري مقارنةُ كل زوجٍ من العناصر فيما بينها مرةً واحدةً على الأكثر. لماذا؟ لأن العناصرَ تُقارَن بالعنصر المحوري فقط، وبعد انتهاء استدعاءٍ محدَّدٍ لإجرائية PARTITION لا يقارَن العنصرُ المحوري المستخدَم في هذا الاستدعاء بأيِّ عنصرٍ آخر أبدًا.

يَستخدم تحليلنا متحولات عشوائية مؤشرة indicator random variables (انظر المقطع 2.5). نُعرِّف:

$$X_{ij} = \mathrm{I}\{z_i \text{ is compared to } z_j\} ,$$

حيث نعتبر المقارنات التي تجري في أي وقت خلال تنفيذ الخوارزمية، وليس خلال تكرارٍ واحد أو استدعاءٍ واحد لإجراء PARTITION فقط. ولما كانت مقارنة كل زوج تجري مرةً واحدة على الأكثر، يمكننا بسهولة تقدير العدد الكلي للمقارنات في الخوارزمية:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} .$$

وبأخذ توقُّع الطرفين، ثم باستخدام خطية التوقع والتوطئة 1.5، نحصل على:

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] \end{aligned}$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} \tag{2.7}$$

يبقى علينا حساب $\Pr\{z_i \text{ is compared to } z_j\}$؛ أي احتمال مقارنة z_i بـ z_j. يفترض تحليلنا أن إجراء RANDOMIZED-PARTITION يختار كلَّ محورٍ عشوائيًّا واستقلاليًّا.

لندرس الحالات التي لا تَجري فيها المقارنة بين عنصرين. لنأخذ دخلاً للفرز السريع الأعداد من 1 إلى 10 (بأي ترتيب كان)، ولنفترض أن أول عنصرٍ محوريٍّ هو 7. إن أول استدعاءٍ لإجراء PARTITION يَفصلُ الأعداد إلى مجموعتين: $\{1,2,3,4,5,6\}$ و $\{8,9,10\}$. ثم تجري مقارنة العنصر 7 بجميع العناصر الأخرى، ولكن لَمْ تجرِ (ولن تجري) مقارنةُ أيِّ عددٍ من المجموعة الأولى (وليكن 2 مثلاً) بأيِّ عددٍ من المجموعة الثانية (وليكن 9 مثلاً).

ولما كنا نفترض أن قيمَ العناصر متمايزةٌ بوجهٍ عام، فإننا نَعْرِف، بعد اختيار المحور x حيث $z_i < x < z_j$، أنه لا يمكن مقارنة z_i و z_j في أي وقت لاحق. من جهة أخرى، إذا اختير z_i محورًا قبل أيِّ عنصرٍ في Z_{ij}، فإن z_i سيقارَن بكلِّ عنصرٍ في Z_{ij}، ما عداه هو نفسه. وبالمثل، إذا اختير z_j محورًا قبل أيِّ عنصر في Z_{ij}، فإن z_j سيقارَن بكلِّ عنصرٍ في Z_{ij}، ما عداه هو نفسه. في مثالنا، تجري مقارنة القيمتين 7 و 9، لأن 7 هو العنصر الأول الذي سيجري اختياره محورًا من $Z_{7,9}$. وبالمقابل، لن تجري مقارنة 2 و 9 أبدًا، لأن أول محورٍ اختير من $Z_{2,9}$ هو 7. وهكذا، تجري مقارنة z_i و z_j إذا وفقط إذا كان العنصر الأول الذي سيجري اختياره محورًا من Z_{ij} هو إما z_i وإما z_j.

نحسب الآن احتمال وقوع هذا الحدث. إن المجموعة Z_{ij} تكون بكاملها في الجزء نفسه، قبل اللحظة التي نختار فيها أحد عناصر Z_{ij} ليكون محورًا. ولذا، فإن أيَّ عنصرٍ من Z_{ij} يمكن أن يكون أولَّ عنصرٍ يُختار محورًا بالاحتمال نفسه. ولما كانت المجموعة Z_{ij} تحوي $j - i + 1$ عنصرًا، وكانت المحاور مختارةً بعشوائيةٍ واستقلالية، فإن احتمال أن يكون عنصرٌ ما هو العنصر الأول المختار ليكون محورًا هو $1/(j - i + 1)$. وبذلك، يكون لدينا[1]:

$$\begin{aligned}\Pr\{z_i \text{ is compared to } z_j\} &= \Pr\{z_i \text{ or } z_j \text{ is first pivot chosen from } Z_{ij}\}\\ &= \Pr\{z_i \text{ is first pivot chosen from } Z_{ij}\}\\ &\quad + \Pr\{z_j \text{ is first pivot chosen from } Z_{ij}\}\end{aligned}$$

[1] أي إن احتمال أن يقارن z_i بـ z_j يساوي احتمال أن يكون z_i أو z_j هو أول محور جرى اختياره من Z_{ij}، وهذا يساوي مجموع احتمال أن يكون z_i هو أول محور جرى اختياره من Z_{ij} واحتمال أن يكون z_i هو أول محور جرى اختياره من Z_{ij}.

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

يمكن الانتقال من السطر الأول إلى السطر الثاني لأن الحدثَيْن يُقصي أحدهما الآخر **mutually exclusive**. وبدمج المعادلتين (2.7) و (3.7)، نجد أن

$$\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

يمكننا أن نقيّم هذا المجموع باستخدام تبديل المتحولات $(k = j - i)$ والحد على السلاسل التوافقية harmonic series في المعادلة (أ.7):

$$\begin{aligned} \mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k+1} \\ &< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\ &= \sum_{i=1}^{n-1} O(\lg n) \\ &= O(n \lg n) \ . \end{aligned} \tag{4.7}$$

نستنتج من ذلك أنه عند استخدام RANDOMIZED-PARTITION، يكون زمن التنفيذ المتوقع للفرز السريع عندما تكون قيم العناصر متمايزة هو $O(n \lg n)$.

تمارين

1-4.7

بيّن أن المعادلة التكرارية $T(n) = \max_{0 \le q \le n-1}(T(q) + T(n-q-1)) + \Theta(n)$ تحقق $T(n) = \Omega(n^2)$.

2-4.7

بيّن أن زمن تنفيذ الفرز السريع في أحسن الحالات هو $\Omega(n \lg n)$.

3-4.7

بيّن أن التعبير $q^2 + (n-q-1)^2$ يصل إلى قيمته العظمى ضمن $q = 0,1, \ldots, n-1$ عندما

تكون $q = 0$ أو $q = n - 1$.

4-4.7

بيّن أن زمن التنفيذ المتوقع لإجراء RANDOMIZED-QUICKSORT هو $\Omega(n \lg n)$.

5-4.7

يمكن تحسين زمن تنفيذ الفرز السريع عمليًّا بالاستفادة من زمن التنفيذ السريع للفرز بالإدراج حين يكون دخله مفروزًا "تقريبيًا". عند استدعاء الفرز السريع على صفيفة جزئية عدد عناصرها أقل من k عنصرًا، دَعْهُ يَعُدْ return دون فرز الصفيفة الجزئية. وبعد أن يعود استدعاءُ الفرز السريع ذو المستوى الأعلى، نفِّذْ الفرز بالإدراج على كامل الصفيفة لإنهاء إجرائية الفرز. برهن أن خوارزمية الفرز هذه تُنفَّذ في زمن متوقع $O(nk + n \lg(n/k))$. كيف يجب انتقاء k، من الناحيتين النظرية والعملية؟

★ *6-4.7*

لنبحث في تعديل إجراء PARTITION بانتقاء عشوائي لثلاثة عناصر من الصفيفة A، ثم بالتجزئة حول وسطها median (القيمة الوسطى للعناصر الثلاثة). قرِّبْ احتمال الحصول على تفريق بنسبةِ α إلى $(1 - \alpha)$ في أسوأ الحالات، وذلك بصيغة دالة بدلالة α على المجال $0 < \alpha < 1$.

مسائل

1-7 صحة تجزئة Hoare

إن نسخة PARTITION المعطاة في هذا الفصل ليست نسخة الخوارزمية الأصلية للتجزئة. وفيما يلي الخوارزمية الأصلية للتجزئة، وتُنسَب إلى C. A. R. Hoare:

```
HOARE-PARTITION(A, p, r)
 1  x = A[p]
 2  i = p - 1
 3  j = r + 1
 4  while TRUE
 5      repeat
 6          j = j - 1
 7      until A[j] ≤ x
 8      repeat
 9          i = i + 1
10      until A[i] ≥ x
11      if i < j
12          exchange A[i] with A[j]
13      else return j
```

أ. اعرض ناتج تطبيق HOARE-PARTITION على الصفيفة

$A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 \rangle$

بحيث تَظهر قيم الصفيفة والقيم المساندة بعد كل تكرار لحلقة **while** في الأسطر 4-13.

يطلب إليك في الأسئلة الثلاثة التالية أن تقدم دليلاً محكمًا على أن إجراء HOARE-PARTITION صحيح. بافتراض أن الصفيفة الجزئية $A[p\,..\,r]$ تتضمن عنصرين على الأقل، أثبت ما يلي:

ب. يحقق الدليلان i و j عدم إمكان الوصول إلى أي عنصر من A واقع خارج الصفيفة الجزئية $A[p\,..\,r]$.

ت. عندما تنتهي HOARE-PARTITION، تعيد قيمة j بحيث تحقق $p \leq j < r$.

ث. عندما تنتهي HOARE-PARTITION يكون أي عنصر من $A[p\,..\,j]$ أصغر أو يساوي أي عنصر من $A[j+1\,..\,r]$.

يَفصل إجراء PARTITION في المقطع 1.7 قيمة المحور (الموجودة أصلاً في $A[r]$) عن الجزأين اللذين تشكلهما. من ناحية أخرى، يضع إجراء HOARE-PARTITION قيمةَ المحور (الموجودة أصلاً في $A[p]$) في أحد الجزأين $A[p\,..\,j]$ و $A[j+1\,..\,r]$ دومًا. ولما كانت $p \leq j < r$، فهذا التفريق ليس بديهياً دومًا.

ج. أعد كتابة إجراء QUICKSORT بحيث يستخدم HOARE-PARTITION.

2-7 الفرز السريع في حالة تساوي قيم العناصر

يَفترض تحليل الزمن المتوقع للفرز السريع ذي العشوائية المضافة في المقطع 7.4-2 أن قيم جميع العناصر متمايزة. ندرس في هذه المسألة ماذا يحدث عندما لا تكون كذلك.

أ. افترض أن قيم جميع العناصر متساوية. ماذا يمكن أن يكون زمن تنفيذ الفرز السريع ذي العشوائية المضافة في هذه الحالة.

ب. يعيد إجراء PARTITION دليلاً q بحيث أن قيمة كل عنصر في $[p\,..\,q-1]$ أصغر أو تساوي $A[q]$ وقيمة كل عنصر في $A[q+1\,..\,r]$ أكبر من $A[q]$. عدِّل إجراء PARTITION لإنشاء إجراء $\text{PARTITION}'(A, p, r)$، الذي يبدل مواقع عناصر $A[p\,..\,r]$ ويعيد دليلين q و t، حيث $p \leq q \leq t \leq r$، وبحيث تكون

- جميع عناصر $A[q\,..\,t]$ متساوية،
- قيمة كل عنصر في $A[p\,..\,q-1]$ أصغر من $A[q]$،
- وقيمة كل عنصر في $A[t+1\,..\,r]$ أكبر من $A[q]$.

وكما في PARTITION يجب أن يستغرق تنفيذ إجراء PARTITION′ زمنًا $\Theta(r-p)$.

ت. عدّل إجراء RANDOMIZED-PARTITION بحيث يستدعي PARTITION′، وسمِّ الإجراء الجديد RANDOMIZED-PARTITION′. ثم عدِّل إجراء QUICKSORT لإنشاء إجراء $\text{QUICKSORT}'(A,p,r)$ يستدعي RANDOMIZED-PARTITION′ ويطبق عوديًا فقط على الأجزاء التي لا تُعرف عناصرها بأنها متساوية فيما بينها.

ث. باستخدام QUICKSORT′، كيف يمكنك مواءمة التحليل في المقطع 7.4-2 لتجنُّب افتراض أن كل العناصر متمايزة؟

7-3 *تحليل بديل للفرز السريع*

يركز تحليلٌ بديلٌ لزمن تنفيذ الفرز السريع ذي العشوائية المضافة على زمن التنفيذ المتوقع لكل استدعاء عودي مستقل لإجراء RANDOMOZED-QUICKSORT، بدلاً من عدد المقارنات المنفذة.

أ. ناقش أنه إذا كانت لدينا صفيفة طولها n، فإن احتمال أن يجري اختيار عنصر ما محورًا هو $1/n$. استخدم هذا لتعريف متحولات عشوائية مؤشرة $X_i = I$ {يجري اختيار العنصر ذي الترتيب i من حيث الصغر محورًا}. ما هو $\text{E}[X_i]$؟

ب. ليكن $T(n)$ متحولاً عشوائيًا يرمز إلى زمن تنفيذ الفرز السريع على صفيفة طولها n. ناقش أن :

$$\text{E}[T(n)] = \text{E}\left[\sum_{q=1}^{n} X_q\big(T(q-1)+T(n-q)+\Theta(n)\big)\right] . \tag{5.7}$$

ت. بيّن أن بالإمكان كتابة المعادلة (5.7) على الشكل:

$$\text{E}[T(n)] = \frac{2}{n}\sum_{q=2}^{n-1}\text{E}[T(q)] + \Theta(n) \tag{6.7}$$

ث. بيّن أن:

$$\sum_{k=2}^{n-1} k\lg k \le \frac{1}{2}n^2\lg n - \frac{1}{8}n^2 \tag{7.7}$$

(*تلميح*: افصل المجموع إلى جزأين، الأول لقيم $k = 2,3,\dots,\lceil n/2\rceil - 1$ والآخر لقيم $k = \lceil n/2\rceil,\dots,n-1$).

ج. بيّن باستخدام الحد الموجود في المعادلة (7.7) أن للمعادلة التكرارية (6.7) حلاً $\text{E}[T(n)] = \Theta(n\lg n)$. (*تلميح*: أثبت بالتعويض، أنَّ $\text{E}[T(n)] \le an\lg n$ في حالة كون n كبيرة كفاية و a ثابتًا موجبًا.)

4-7 عمق المكدس في الفرز السريع

تتضمن خوارزمية QUICKSORT في المقطع 1.7 استدعاءَيْن عوديين لنفسها. بعد استدعاء QUICKSORT لإجراء PARTITION يَفرز عوديًّا الصفيفةَ الجزئية اليسرى ثم الصفيفة الجزئية اليمنى. يعتبر الاستدعاء العودي الثاني لخوارزمية QUICKSORT غيرَ ضروري حقًّا؛ إذ يمكن تجنبه باستخدام بنية تحكم تكرارية. تتوفَّر هذه التقنية، التي تسمى ***عَوْدية الذيل tail recursion***، آليًّا في المترجمات compilers الجيدة. لنأخذ النسخة التالية من الفرز السريع، الذي يحاكي عودية الذيل.

```
TAIL-RECURSIVE-QUICKSORT(A, p, r)
1  while p < r
2      // Partition and sort left subarray
3      q = PARTITION(A, p, r)
4      TAIL-RECURSIVE-QUICKSORT'(A, p, q − 1)
5      p = q + 1
```

أ. ناقش أن TAIL-RECURSIVE-QUICKSORT$(A, 1, A.length)$ تفرز الصفيفة A فرزًا صحيحًا.

تنفِّذ المترجماتُ عادةً الإجرائياتِ العوديةَ باستخدام ***مكدسٍ Stack*** يحوي معلوماتٍ متعلقةً بكلِّ استدعاءٍ عَوْدي، ومن ضمنها قيمُ المُوسِطات parameter values. توجد المعلوماتُ المتعلقة بأحدث استدعاء على قمة المكدس، والمعلوماتُ المتعلقة بالاستدعاء الابتدائي في الأسفل. عند طلب الإجراء، يجري ***دفع*** معلوماته في المكدس؛ وعندما تنتهي، ***تُنزع*** معلوماته من المكدس. ولما كنا نفترض أن موسطات الصفيفة تُمثَّلُ بمؤشرات، فإن المعلومات المتعلقة بكل استدعاء إجراء على المكدس تتطلب حجم تخزين $O(1)$ في المكدس. يُعرَّف ***عمق المكدس Stack depth*** بأنه مقدار حجم التخزين الأعظمي المستخدَم في أي وقت أثناء الحساب.

ب. صِفْ مشهدًا يكون فيه عمق المكدس في TAIL-RECURSIVE-QUICKSORT $\Theta(n)$ على صفيفة دخل ذات n عنصرًا.

ت. عدّل رماز TAIL-RECURSIVE-QUICKSORT بحيث يكون عمق المكدس في أسوأ الحالات $\Theta(\lg n)$. حافظ على زمن تنفيذ متوقع للخوارزمية $O(n \lg n)$.

5-7 التجزئة وفق وسط ثلاثة عناصر

إحدى طرائق تحسين إجراء RANDOMIZED-QUICKSORT هي إجراء التجزئة حول محور يجري اختياره بعناية أكبر من مجرد أخذ عنصر عشوائيًّا من الصفيفة الجزئية. إحدى المنهجيات الشائعة هي طريقة ***وسط الثلاثة median-of-3***: اختر المحور هو العنصر الوسط median (العنصر الأوسط middle) من مجموعة مؤلفة من 3 عناصر جرى اختيارها عشوائيًّا من الصفيفة الجزئية. (انظر التمرين 7.4-6.) لنفترض، في هذه المسألة، أن

عناصر صفيفة الدخل $A[1\,..\,n]$ متمايزة وأن $n \geq 3$. نرمز لصفيفة الخرج المفروزة بالرمز $A'[1\,..\,n]$. عرِّف، باستخدام طريقة وسط الثلاثة لاختيار المحور x، القيمةَ $p_i = \Pr\{x = A'[i]\}$.

أ. أعط الصيغة الدقيقة لقيم p_i كدالة في n و i لقيم $i = 2,3,\ldots,n-1$. (لاحظ أن $p_1 = p_n = 0$.)

ب. بكم زدنا إمكان likelihood كون المحور $x = A'[\lfloor(n+1)/2\rfloor]$ هو وسط الصفيفة $A[1\,..\,n]$، مقارنةً بالتنجيز العادي؟ افترض أن $n \to \infty$، وأعطِ نسبة حدية لهذا الاحتمال.

ت. إذا عرّفنا التفريق "الجيد" بأنه اختيار المحور $x = A'[i]$، حيث $n/3 \leq i \leq 2n/3$، بكم نكون قد زدنا إمكان الحصول على تفريق جيد مقارنةً بالتنجيز العادي؟ (*تلميح*: قرِّب المجموع إلى تكامل.)

ث. ناقش كون طريقة وسط الثلاثة تؤثر فقط على العامل الثابت في زمن تنفيذ الفرز السريع $\Omega(n\lg n)$.

6-7 *الفرز الترجيحي للمجالات*

لنأخذ مسألةَ فرزٍ لا نَعرف فيها الأعدادَ بالتحديد. بل نَعرف أن لكلِّ عددٍ مجالاً ينتمي إليه على مستقيم الأعداد الحقيقية. أي لدينا n مجالاً مغلقًا من الشكل $[a_i, b_i]$، حيث $a_i \leq b_i$. الهدف هنا هو فرز هذه المجالات ***فرزًا ترجيحيًا*** ***fuzzy-sort***، أي إيجاد تبديل permutation $\langle i_1, i_2, \ldots, i_n\rangle$ من المجالات بحيث أنه في حالة $j = 1,2,\ldots,n$ يوجد $c_j \in [a_{i_j}, b_{i_j}]$ يحقِّق $c_1 \leq c_2 \leq \cdots \leq c_n$.

أ. صمِّم خوارزميةً ذات عشوائية مضافة لفرز n مجالاً ترجيحيًا. يجب أن يكون لخوارزميتك البنية العامة لخوارزمية تفرز سريعًا النقاط الحدية اليسرى (قيم a_i)، ولكنها يجب أن تستفيد من المجالات المتراكبة overlapping لتحسين زمن التنفيذ. (كلما تراكبت المجالات أكثر، أصبحت مسألة فرز المجالات ترجيحيًا أسهل. يجب أن تستفيد خوارزميتك من هذا التراكب إلى أقصى حد.)

ب. برهن أن خوارزميتك هذه تُنفَّذُ بزمنٍ متوقع $\Theta(n\lg n)$ في الحالة العامة، ولكنها تُنفَّذُ في زمن متوقع $\Theta(n)$ عندما تتراكب جميع المجالات (أي عندما توجد قيمة x بحيث يكون $x \in [a_i, b_i]$ لكل قيم i). يجب ألا تتحقق خوارزميتك من هذه الحالة بصورة صريحة، بل يجب أن يتحسن أداؤها طبيعيًا عندما يزداد حجم التراكب.

ملاحظات الفصل

ابتكر Hoare [170] إجراء الفرز السريع؛ وقد عرضنا نسخته في المسألة 1.7. ويعود الفضل في إجراء PARTITION المذكور في المقطع 1.7 إلى N. Lomuto. ويعود التحليل في المقطع 4.7 إلى Avrim Blum.

يقدِّم Sedgewick [305] و Bentley [43] مرجعًا جيدًا حول تفاصيل التنجيز ومدى أهميتها.

بيّن McIlroy [248] كيفية هندسة "خصم قاتل killer adversary" يولِّد صفيفةً يَستغرق أيُّ تنجيزٍ للفرز السريع عليها، افتراضيًّا، زمنًا $\Theta(n^2)$. إذا كان التنجيزُ ذا عشوائية مضافة، فإن الخصم يولِّد الصفيفةَ بعد مشاهدة الخيارات العشوائية من خوارزمية الفرز السريع.

8 الفرز في زمن خطي

عرضنا حتى الآن العديدَ من الخوارزميات التي تستطيع فرز n عددًا في زمن $O(n \lg n)$. يُحرِز الفرزُ بالدمج والفرز بالكومة هذا الحد الأعلى في أسوأ الحالات؛ في حين يُحرِز الفرز السريع هذا الحد على نحو وسطي. إضافةً إلى ذلك، يمكننا، في كل خوارزمية من هذه الخوارزميات، إيجاد متتالية من n عددًا تؤدي إلى تنفيذ الخوارزمية في زمن $\Omega(n \lg n)$.

تتشارك هذه الخوارزميات بخاصية جديرة بالاهتمام: *يعتمد الترتيب المفروز الذي تحدده هذه الخوارزميات فقط على المقارنات بين عناصر الدخل.* نُسمي مثل خوارزميات الفرز هذه بـ ***الفرز بالمقارنة comparison sorts***. إن جميع خوارزميات الفرز المقدَّمة حتى الآن هي من نوع الفرز بالمقارنة.

نبرهن في المقطع 1.8 أن أية خوارزمية فرز بالمقارنة يجب أن تُجري $\Omega(n \lg n)$ عملية مقارنة في أسوأ الحالات لفرز n عنصرًا. وعلى ذلك، فإن الفرز بالدمج والفرز بالكومة أمثليَّيْن بالمقارنة، ولا توجد خوارزميات فرز بالمقارنة أسرع منهما بأكثر من معامل ثابت.

تناقش المقاطع 2.8 و 3.8 و 4.8 ثلاث خوارزميات فرز تُنفَّذ في زمن خطي؛ وهي: الفرز بالعدّ counting sort، والفرز حسب الأساس radix sort، والفرز بالدِّلاء bucket sort. تَستخدم هذه الخوارزميات، طبعًا، عملياتٍ غير عمليات المقارنة لتحديد الترتيب المفروز. لذلك، فإن الحد الأدنى $\Omega(n \lg n)$ لا ينطبق عليها.

1.8 الحدود الدنيا للفرز

لا نُجري في الفرز بالمقارنة سوى مقارنات بين العناصر، وذلك للحصول على معلومات عن ترتيب متتالية دخل $\langle a_1, a_2, \ldots, a_n \rangle$. أي، إذا كان لدينا العنصران a_i و a_j، فإننا ننجز أحد الاختبارات: $a_i < a_j$ أو $a_i \le a_j$ أو $a_i = a_j$ أو $a_i \ge a_j$ أو $a_i > a_j$ لتحديد ترتيبهما النسبي. ولا يمكننا فحص قيم العناصر أو الحصول على معلومات عن ترتيبها بأية طريقة أخرى.

سنفترض في هذا المقطع، دون أن يؤثر ذلك على العمومية، أن جميع عناصر الدخل متمايزة. إذا أخذنا

هذه الفرضية بالحسبان، فإن مقارنات من الشكل $a_i = a_j$ ستكون عديمة الجدوى، لذا يمكننا أن نفترض أنه لن تُجرى مقارنات من هذا الشكل. نُشير أيضًا إلى أن المقارنات $a_i \leq a_j$ و $a_i \geq a_j$ و $a_i > a_j$ و $a_i < a_j$ جميعها متكافئة لكونها تقدم معلومات متطابقة عن الترتيب النسبي لـ a_i و a_j. لذلك، فإننا سنفترض أن جميع المقارنات هي من الشكل $a_i \leq a_j$.

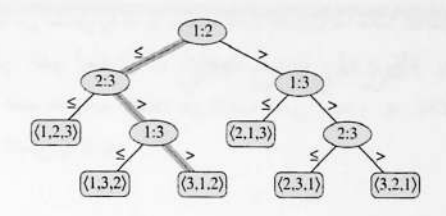
نموذج شجرة القرار

يمكننا دراسة الفرز بالمقارنة على نحو مجرد باستخدام أشجار القرار. ***شجرة القرار decision trees*** هي شجرة ثنائية ملأى تُمثِّل المقارنات بين العناصر التي تُنجَّز باستخدام خوارزمية فرزٍ محدَّدة تعمل على دخلٍ حجمُه معطى. يتجاهل هذا التمثيل عمليات التحكم وتحريك المعطيات وجميع الجوانب الأخرى للخوارزمية. يبين الشكل 1.8 شجرة قرار مقابلة لخوارزمية الفرز بالإدراج، المشروحة في المقطع 1.2، تعمل على متتالية دخل من ثلاثة عناصر.

نُشير، في شجرة القرار، إلى كلِّ عقدة داخلية بـ $i : j$ لقيمةٍ ما لـ i و j ضمن المجال $1 \leq i, j \leq n$، حيث n عدد العناصر في متتالية الدخل. ونُشير أيضًا إلى كل ورقة بتبديل $\langle \pi(1), \pi(2), \ldots \pi(n) \rangle$. (انظر المقطع ت.1 للاطلاع على التباديل.) يقابِل تنفيذُ خوارزمية الفرز تعقُّبَ مسارٍ بسيط ما من جذر شجرة القرار نزولاً إلى أحد الأوراق. تُشير كل عقدة داخلية إلى المقارنة $a_i \leq a_j$. تُملي بعدها الشجرة الجزئية اليسرى مقارنات تالية عندما $a_i \leq a_j$، في حين تُملي الشجرة الجزئية اليمنى مقارنات تالية عندما $a_i > a_j$. عندما نصل إلى ورقةٍ ما، تكون خوارزمية الفرز قد أرسَت الترتيب $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$. وحيث إنه يجب أن تكون أيةُ خوارزميةِ فرزٍ صحيحةٍ قادرةً على إنتاج كلِّ تبديلٍ من تباديل دَخْلها، فإن كلَّ تبديلٍ من تباديل n عنصرًا (وعددها $n!$) يجب أن يَظهر على أنه إحدى أوراق شجرة القرار حتى يكون الفرزُ بالمقارنة صحيحًا. إضافةً إلى ذلك، يجب أن يكون من الممكن الوصول إلى كلٍّ من هذه الأوراق، بدءًا من الجذر عبر مسارٍ نازلٍ مقابلٍ لتنفيذٍ فعليٍّ للفرز بالمقارنة. (نَصِفُ مثل هذه الأوراق بأنها "أوراق يمكن الوصول إليها reachable".) لذا، فإننا سنقصر دراستنا على أشجار القرار التي يَظهر فيها كلُّ تبديلٍ على أنه ورقةٌ يمكن الوصول إليها.

حد أدنى لأسوأ الحالات

إن طول أطول مسارٍ بسيطٍ من جذر شجرة قرارٍ إلى أيٍّ من أوراقها التي يمكن الوصول إليها يُمثِّل عدد المقارنات التي تنفِّذها خوارزميةُ البحث المقابلة في أسوأ الحالات. لذلك، فإن عدد المقارنات في أسوأ الحالات لخوارزميةِ فرزٍ بالمقارنة يساوي ارتفاعَ شجرة قرارها. إن حدًّا أدنى لارتفاعات جميع أشجار القرار التي يظهر فيها كل تبديل على أنه ورقةٌ يمكن الوصول إليها هو إذن حدٌّ أدنى لزمن تنفيذ أية خوارزمية بحثٍ بالمقارنة. تُعطي المبرهنة التالية مثل هذا الحد الأدنى.



الشكل 1.8 شجرة القرار لفرزٍ بالإدراج يعمل على ثلاثة عناصر. تشير العقدة الداخلية المشار إليها بـ $i:j$ إلى مقارنةٍ بين a_i و a_j. وتشير الورقة المشار إليها بالتبديل $\langle\pi(1),\pi(2),\dots,\pi(n)\rangle$ إلى الترتيب $a_{\pi(1)} \le a_{\pi(2)} \le \cdots \le a_{\pi(n)}$. ويُشير المسارُ المظلَّل إلى القرارات المتخَذة عند فرز متتالية الدخل $\langle a_1 = 6, a_2 = 8, a_3 = 5\rangle$؛ ويشير التبديل $\langle 3,1,2\rangle$ عند الورقة إلى أن الترتيب المفروز هو $a_3 = 5 \le a_1 = 6 \le a_2 = 8$. لدينا $3! = 6$ تباديل ممكنة لعناصر الدخل، لذلك يجب أن تحتوي شجرة القرار على الأقل على 6 أوراق.

مُبرهنة 1.8

تحتاج أية خوارزمية بحث بالمقارنة إلى $\Omega(n\lg n)$ عملية مقارنة في أسوأ الحالات.

البرهان يكفي، من المناقشة السابقة، تحديدُ ارتفاع شجرة القرار التي يَظهر فيها كل تبديل على أنه ورقةٌ يمكن الوصول إليها. لتكن لدينا شجرة قرار ارتفاعها h ولها l ورقةً يمكن الوصول إليها، وهي شجرة تقابل فرزًا بالمقارنة على n عنصرًا. ولما كان كلُّ تبديل من تباديل الدخل (وعددها $n!$) يَظهر على أنه إحدى الأوراق، فإن $n! \le l$. ولما كانت أيةُ شجرةٍ ثنائيةٍ ارتفاعها h، لا تزيد عدد أوراقها على 2^h ورقة، فإن لدينا

$$n! \le l \le 2^h \ ,$$

وبأخذ لغاريتم الطرفين نجد:

$$h > \lg(n!) \qquad \text{(لأن الدالة lg متزايدة بانتظام)}$$

$$> \Omega(n\lg n) \qquad \text{((من المعادلة (19.3))}$$ ■

نتيجة 2.8

الفرز بالكومة والفرز بالدمج هما خوارزميتا فرزٍ بالمقارنة أمثليتان بالمقاربة.

البرهان إن الحدود العليا $O(n\lg n)$ لأزمنة تنفيذ الفرز بالكومة والفرز بالدمج تطابق الحد الأدنى في أسوأ الحالات $\Omega(n\lg n)$ من المبرهنة 1.8. ■

تمارين

1-1.8

ما هو أصغر عمق ممكن لورقة في شجرة قرار الفرز بالمقارنة؟

2-1.8

احصل على حد ملاصق بالمقارنة لـ $\lg(n!)$ دون استخدام تقريب ستيرلينغ. وذلك بأن تُقدِّرَ المجموع $\sum_{k=1}^{n} \lg k$ باستخدام تقنيات من المقطع أ.2.

3-1.8

بيِّنْ أنه لا توجد خوارزميةُ فرزٍ بالمقارنة زمنُ تنفيذها خطيٌّ لنصف المُدخلات (التي عددها $n!$ وطولها n) على الأقل. ماذا عن الجزء $1/n$ من المدخلات التي طولها n؟ وماذا عن الجزء $1/2^n$؟

4-1.8

ليكن المطلوب فرز متتالية من n عنصرًا. تتكون متتالية الدخل من n/k متتاليةً جزئية، تحتوي كلٌّ منها على k عنصرًا. إن جميع العناصر في متتالية جزئية ما أصغر من العناصر في المتتالية الجزئية التالية وأكبر من العناصر في المتتالية الجزئية السابقة. ومن ثَم، فإن كل ما نحتاج إليه لفرز كامل المتتالية بطول n هو فرز k عنصرًا لجميع الـ n/k متتالية جزئية. أعطِ حدًّا أدنى من الرتبة $\Omega(n \lg k)$ لعدد المقارنات اللازمة لحلّ هذا الصيغة المعدَّلة من مسألة الفرز. (*تلميح*: إن تجميع الحدود الدنيا لكل من المتتاليات الجزئية تنقصه الدقة الرياضية.)

2.8 الفرز بالعد

يفترض ***الفرز بالعد counting sort*** أنَّ كلَّ عنصرٍ من عناصر الدخل (n عنصرًا) هو عددٌ صحيح من المجال من 0 إلى k، حيث k عدد صحيح. فإذا كان $k = O(n)$، فإن الفرز يُنَفَّذ في زمن $\Theta(n)$.

يُحدِّد الفرزُ بالعدِّ، لكلِّ عنصرٍ دخلٍ x، عددَ العناصر التي هي أصغر من x. يَستخدم الفرزُ هذه المعلومات لوضع العنصر x مباشرةً في موقعه في صفيفة الخرج. فمثلاً، إذا وُجِدَ 17 عنصرًا أقل من x، فإن x تَظهر في الخرج في الموقع 18. وعلينا تعديل هذه الآلية قليلاً لمعالجة الحالة التي يوجد فيها عدة عناصر لها القيمة نفسها، لأننا لا نريد وضعها جميعًا في الموقع نفسه.

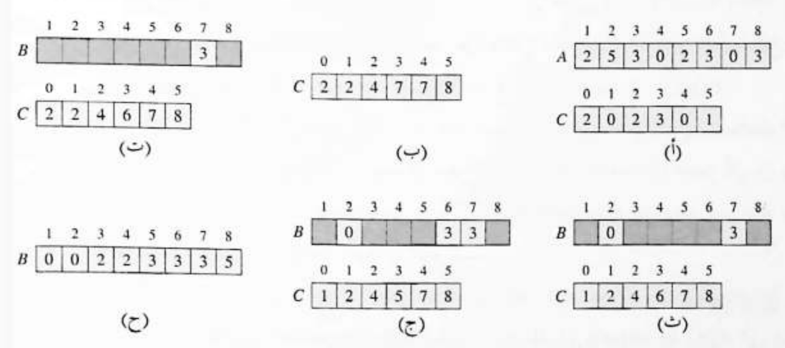
نفترض، في رماز الفرز بالعد، أن الدخلَ هو الصفيفة $A[1..n]$، ومن ثَم فإن $A.length = n$. نحتاج إلى صفيفتين إضافيتين: الصفيفة $B[1..n]$ لتخزين الخرج المفروز، والصفيفة $C[0..k]$ التي تتيح ذاكرة تخزين مؤقتة.

```
COUNTING-SORT(A, B, k)
1  let C[0..k] be a new array
2  for i = 0 to k
```

```
3        C[i] = 0
4    for j = 1 to A.length
5        C[A[j]] = C[A[j]] + 1
6    // C[i] now contains the number of elements equal to i.
7    for i = 1 to k
8        C[i] = C[i] + C[i − 1]
9    // C[i] now contains the number of elements less than or equal to i.
10   for j = A.length downto 1
11       B[C[A[j]]] = A[j]
12       C[A[j]] = C[A[j]] − 1
```

يوضح الشكل 2.8 الفرز بالعد. بعد حلقة **for** في السطرين 2-3 التي تسند أصفارًا إلى جميع عناصر الصفيفة C، تتفحص حلقة **for** في السطرين 4-5 قيمة كل عنصر دخل. إذا كانت قيمة عنصر دخل تساوي i، فإننا نزيد واحدًا على $C[i]$. وهكذا، تحتفظ $C[i]$، بعد السطر 5، بعدد عناصر الدخل التي تساوي i لكل عدد صحيح $i = 0, 1, \dots, k$. يُحدِّد السطران 7-8، لكل $i = 0, 1, \dots, k$، عدد عناصر الدخل التي هي أقل من (أو تساوي) i وذلك بالقيام بجمع تراكمي running sum للصفيفة C.

في النهاية، تضع حلقة **for** في الأسطر 10-12 كلَّ عنصر $A[j]$ في مكان فرزه الصحيح ضمن الصفيفة B. إذا كانت جميعُ العناصر (وعددها n) متمايزةً، فإننا عندما نُدخل السطر 10 أوَّلَ مرةٍ، لكلِّ عنصر $A[j]$، فإن قيمةَ $C[A[j]]$ تعطي المكانَ النهائي الصحيح لـ $A[j]$، لأنه يوجد $C[A[j]]$ عنصرًا أصغر من (أو تساوي) $A[j]$. ولكن لما كان من الممكن ألا تكون العناصر متمايزة، فإننا نُنقِص $C[A[j]]$ في كل مرة نضع



الشكل 2.8 تطبيق COUNTING-SORT على صفيفة الدخل $A[1..8]$، حيث كل عنصر من A هو عدد صحيح غير سالب لا يزيد على $k = 5$. **(أ)** الصفيفة A والصفيفة المساعدة C بعد السطر 5. **(ب)** الصفيفة C بعد السطر 8. **(ت)**–**(ج)** صفيفة الخرج B والصفيفة المساعدة C بعد تكرار الحلقة في الأسطر 10-12 مرة ومرتين وثلاث مرات، على الترتيب. العناصر المظلَّلة قليلاً فقط من الصفيفة B هي التي جرى تعبئتها. **(ح)** صفيفة الخرج النهائية المرتبة B.

القيمة $A[j]$ في الصفيفة B. يؤدي إنقاص $C[A[j]]$ إلى وضعِ عنصرٍ تالٍ له قيمةَ $A[j]$ نفسها – إن وُجِد هذا العنصر – قبل موقع $A[j]$ تمامًا في صفيفة الخرج.

كم من الوقت يتطلب تنفيذ الفرز بالعد؟ تستغرق حلقة **for** في السطرين 2-3 زمنًا $\Theta(k)$، وفي السطرين 4-5 زمنًا $\Theta(n)$، وفي السطرين 7-8 زمنًا $\Theta(k)$، وفي الأسطر 10-12 زمن $\Theta(n)$. وبذلك، يكون الزمن الكلي $\Theta(k+n)$. عمليًا، نستخدم الفرز بالعد عادةً عندما يكون لدينا $k = O(n)$، ويساوي زمن التنفيذ في هذه الحالة $\Theta(n)$.

يتغلَّب الفرز بالعد على الحد الأدنى $\Omega(n \lg n)$ المبرهن في المقطع 1.8 لكونه ليس فرزًا بالمقارنة. في الواقع، ليست هناك أية مقارنة بين عناصر الدخل في أي مكان من الرماز. عوضًا عن ذلك، يَستخدم الفرزُ بالعد القيمَ الفعلية للعناصر كمؤشرات داخل صفيفة. إن الحد الأدنى $\Omega(n \lg n)$ للفرز لا ينطبق عندما نبتعد عن نموذج الفرز بالمقارنة.

إحدى الخواص الهامة للفرز بالعد هي ***الاستقرار stable***: حيث تَظهر الأعدادُ التي لها القيمة نفسها في صفيفة الخرج بالترتيب نفسه التي تكون عليه في صفيفة الدخل. وهذا يعني، أنه عند تساوي عددين تُعتمَد القاعدة التي تنصُّ على أن العددَ الذي يَظهر أولاً في صفيفة الدخل يَظهر أولاً في صفيفة الخرج. هذا وتتجلَّى أهميةُ خاصية الاستقرار، عادةً، عندما يكون هناك معطيات تابعة للعنصر الذي يجري فرزه. وثمة سببٌ آخرُ يتعلَّق بأهمية استقرار الفرز بالعد، وهو أن الفرز بالعد كثيرًا ما يُستخدم باعتباره مساقًا فرعيًا في الفرز حسب الأساس radix sort. وسنرى في المقطع التالي، أن الفرز بالعد يجب أن يكون مستقرًا كي يكون عمل الفرز حسب الأساس صحيحًا.

تمارين

1-2.8

اشرح، بالاستعانة بالشكل 2.8 نموذجًا، عَمَلَ COUNTING-SORT على الصفيفة

$A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$

2-2.8

برهن أن COUNTING-SORT مستقر.

3-2.8

افترض أننا أعدنا كتابة حلقة **for** التي تبدأ في السطر 10 لإجرائية COUNTING-SORT كما يلي:

10 **for** $j = 1$ to $A.length$

بيِّنْ أن الخوارزميةَ سوف تبقى تعمل كما يجب. هل الخوارزمية المُعَدَّلة مستقرة؟

4-2.8

صِفْ، لكلِّ عدد صحيح معطى n ضمن المجال 0 إلى k، خوارزميةً تعالج دَخْلَها سبقيًّا، ثم تجيب عن أيّ

استفسار عن عدد الأعداد الصحيحة من الأعداد n التي تقع ضمن المجال $[a..b]$ في زمن $O(1)$. يجب أن تَستغرق المعالجة السبقيَّة في خوارزميتك زمنًا $\Theta(n+k)$.

3.8 الفرز حسب الأساس

خوارزمية ***الفرز حسب الأساس radix sort*** هي الخوارزمية المستخدمة في آلات فرز البطاقات التي لا نجدها الآن إلا في متاحف الحواسيب. تحتوي كلُّ بطاقة 80 عمودًا، وفي كلِّ عمود، يمكن لآلة تثقيب إحداث ثقبٍ في موقع واحد من 12 موقعًا. يمكن "برمجة" الفارزة ميكانيكيًا بحيث تَفْحص عمودًا محددًا في كل بطاقة من رزمة بطاقات، ثم تضع كلَّ بطاقة في حاوية من 12 حاوية تبعًا لموقع الثقب. بعد ذلك، يَجْمع عاملٌ بطاقات الحاويات حاويةً تلو الأخرى، بحيث تكون البطاقات المثقبة في الموقع الأول في أعلى البطاقات، تليها تلك المثقبة في الموقع الثاني، وهكذا دواليك.

في الأرقام العشرية، يَستخدم كلُّ عمودٍ 10 مواقع فقط. (يُستخدَم الموقعان الآخران لترميز محارف غير رقمية.) وبذلك فإن عددًا مؤلَّفًا من d خانةً يحتلُّ حقلاً مؤلَّفًا من d عمودًا. ولما كان بمقدور فارزة البطاقات النظر إلى عمود واحد فقط في كل مرة، فإن مسألة فرز n بطاقة تمثل أعدادًا من d خانة تحتاج إلى خوارزمية فرز.

من البديهي أنه يمكنك فرز الأعداد تبعًا للخانة الأعلى قيمةً most significant digit، ثم تَفرز كلَّ حاويةٍ من الحاويات الناتجة عوديًا، وبعد ذلك تُجمِّع الرزم بالترتيب. ولما كان يجب وضع البطاقات في تسع حاويات من الحاويات العشر جانبًا لفرز كل حاوية على حده، فإن هذه الإجرائية، لسوء الحظ، تُولِّد العديدَ من كدسات البطاقات الوسيطة التي يجب عليك متابعتها. (انظر التمرين 3.8-5.)

يحل الفرز حسب الأساس مسألة فرز البطاقات بطريقة غير حدسية، وذلك بالفرز أولاً تبعًا للخانة الأصغر قيمة. ثم تَجمع الخوارزميةُ البطاقاتِ في رزمةٍ واحدة، بحيث تَسبق البطاقاتُ في الحاوية 0 البطاقاتِ في الحاوية 1، والتي بدورها تَسبق البطاقاتِ في الحاوية 2، وهكذا ... ثم تُعيد فرز كامل الرزمة مرةً أخرى بحسب الخانة ذات المرتبة الثانية وتُعيد تجميع الحاويات بطريقة مشابهة. تستمر العملية حتى يجري فرز البطاقات تبعًا للخانات d كلها. من المدهش، أنه في هذه المرحلة تكون البطاقات مرتبة كليًّا بحسب الأعداد من d خانة. إذن، يحتاج الفرز فقط إلى d مرورًا على الرزمة. يبين الشكل 3.8 كيف يعمل الفرز حسب الأساس على "رزمة" من 7 أعداد كل منها مؤلَّفٌ من 3 خانات.

وحتى يكون عمل الفرز حسب الأساس صحيحًا، يجب أن يكون فرز الخانات مستقرًّا. إن الفرز الذي تقوم به فارزة البطاقات مستقر، ولكن يجب أن يكون العامل حذرًا حتى لا يغير ترتيب البطاقات عندما يخرجها من الحاوية، حتى لو كانت جميع الأوراق في الحاوية الواحدة لها الرقم نفسه في العمود المختار.

329		720		720		329
457		355		329		355
657		436		436		436
839	⟹	457	⟹	839	⟹	457
436		657		355		657
720		329		457		720
355		839		657		839

الشكل 3.8 عملية الفرز حسب الأساس على لائحةٍ مؤلَّفةٍ من 7 أعداد، وكل عدد يتكون من ثلاث خانات. يبيِّن العمودُ الأيسر لائحةَ الدخل. في حين تبيُّن الأعمدةُ المتبقية اللوائحَ بعد الفرز المتتالي وفق الخانات ذات المراتب المتزايدة. يشير التظليل إلى موقع الخانة التي يجري الفرز عليها فتنتج كلّ لائحةٍ من اللائحة التي تسبقها.

في الحواسيب التقليدية، وهي آلاتٌ ذاتُ نفاذٍ عشوائيٍّ تسلسلي sequential random-access، نَستخدم الفرزَ حسب الأساس أحيانًا لفرز تسجيلات records من المعلومات ذات مفاتيح متعددة الحقول. على سبيل المثال، قد نرغب في فرز تواريخ من ثلاثة مفاتيح: السنة، والشهر، واليوم. يمكننا تنفيذ خوارزمية فرز مع دالة مقارنة بحيث إذا كان لدينا تاريخان فإننا نقارن السنتين، فإذا تساوَتا نقارن الشهرين، فإذا تساوَيا أيضًا نقارن الأيام. بطريقة أخرى، يمكننا فرز المعلومات ثلاث مرات باستخدام فرز مستقر: أولاً نفرز تبعًا للأيام، ثم تبعًا للأشهر، وأخيرًا تبعًا للسنين.

إن رماز الفرز حسب الأساس بسيط. تفترض الإجرائية التالية أن كلَّ عنصرٍ في الصفيفة A ذات الـ n عنصرًا يتكوَّن من d خانة، حيث الخانة 1 هي الخانة الدنيا مرتبةً والخانة d هي العليا مرتبةً.

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      use a stable sort to sort array A on digit i
```

توطئة 3.8

إذا كان لدينا n عددًا، كلٌّ منها يتكوَّن من d خانةً، وكلُّ خانةٍ يمكن أن تأخذ إحدى k قيمةً ممكنة، فإن RADIX-SORT تَفرز هذه الأعداد فرزًا صحيحًا في زمن $\Theta(d(n+k))$ في حال كان الفرز المستقر الذي يستخدمه يستغرق زمنًا $\Theta(n+k)$.

البرهان: نستنتج صحة الفرز حسب الأساس، بالاستقراء، على العمود الذي يجري فرزه (انظر التمرين 3.8-3). يعتمد تحليل زمن التنفيذ على الفرز المستقر المستخدم بوصفه خوارزميةَ فرزٍ وسيطة. عندما تقع كلُّ خانةٍ في المجال من 0 إلى $k-1$ (أي إنها يمكن أن تأخذ إحدى k قيمة ممكنة)، وقيمة k ليست جدُّ كبيرة، فإن الفرز بالعد يكون الخيار البديهي. يَستغرق إذن كل مرورٍ على n عددًا من d خانةً زمنًا $\Theta(n+k)$. ولما كان لدينا d مرورًا، فإن الزمن الكلي للفرز حسب الأساس يساوي $\Theta(d(n+k))$. ■

إذا كان d ثابتًا، وكان $k = O(n)$، يمكننا جعل الفرز حسب الأساس يُنفَّذ في زمن خطي. وفي الحالة الأعم، لدينا بعض المرونة في كيفية تفريق كل مفتاح إلى خانات.

توطئة 4.8

إذا كان لدينا n عددًا، كلٌّ منها يتكوَّن من b بتًّا وأيّ عددٍ صحيح موجب $r \le b$، فإن RADIX-SORT تَفرز هذه الأعداد فرزًا صحيحًا في زمن $\Theta((b/r)(n+2^r))$ إذا كان الفرز المستقر الذي يستخدمه يستغرق زمنًا $\Theta(n+k)$ في حال مُدخلات تقع ضمن المجال من 0 إلى k.

البرهان: يمكننا اعتبار أنَّ كلَّ مفتاحٍ وكأنه يتكوَّن من $d = \lceil b/r \rceil$ خانةً من r بتًّا، لكلِّ قيمة $r \le b$. إنَّ كلَّ خانةٍ هي عددٌ صحيح في المجال من 0 إلى $2^r - 1$، ومنه يمكننا استخدام فرز بالعدِّ لـ $k = 2^r - 1$. (فمثلاً، يمكن اعتبار كلمةٍ من 32 بتًّا، وكأنها مكوَّنةٌ من أربع خاناتٍ كلٌّ منها مؤلَّفٌ من ثمانية بتات. إذن، لدينا $b = 32$، و $r = 8$، و $k = 2^r - 1 = 255$، و $d = b/r = 4$.) يستغرق كل مرور للفرز بالعد زمنًا $\Theta(n+k) = \Theta(n+2^r)$. وحيث إنه لدينا d مرورًا، فإن زمن التنفيذ الكلي يساوي $\Theta(d(n+2^r)) = \Theta((b/r)(n+2^r))$. ■

إذا كانت لدينا قيمتان معطاتان n و b، فكيف نختار القيمة r $r \le b$ التي تجعل التعبير $(b/r)(n+2^r)$ أصغريًّا؟ إذا كانت $b < \lfloor \lg n \rfloor$، فإن $(n+2^r) = \Theta(n)$ لأية قيمةٍ لـ $r \le b$. وهكذا، إذا اخترنا $r = b$ يكون زمن التنفيذ مساويًا $(b/b)(n+2^b) = \Theta(n)$، والتي هي أمثلية بالمقاربة. وإذا كانت $b \ge \lfloor \lg n \rfloor$، فإن اختيار $r = \lfloor \lg n \rfloor$ يحقِّق أفضل زمن ضمن عامل ثابت، وهو ما سنراه هنا. وبذلك، فإن اختيارنا $r = \lfloor \lg n \rfloor$ يجعل زمن التنفيذ من الرتبة $\Theta(bn/\lg n)$. فإذا زدنا r فوق $\lfloor \lg n \rfloor$، فإن الحد 2^r في البسط يزداد بسرعة أكبر من الحد r الموجود في المقام، ومن ثَمّ فإن زيادة r فوق $\lfloor \lg n \rfloor$ يجعل زمن التنفيذ من الرتبة $\Omega(bn/\lg n)$. أما إذا أنقصنا r، عوضًا عن زيادتها، تحت $\lfloor \lg n \rfloor$، فإن الحد b/r سيزداد، ويبقى الحد $n+2^r$ عند $\Theta(n)$.

هل خوارزمية الفرز حسب الأساس أفضل من خوارزميات الفرز التي تعتمد على المقارنة، كالفرز السريع مثلاً؟ إذا كانت $b = O(\lg n)$، كما هو الحال غالبًا، واخترنا $r \approx \lg n$، فإن زمن تنفيذ الفرز حسب الأساس يكون $\Theta(n)$، والذي يَظهر أنه أفضل من توقع زمن الفرز السريع $\Theta(n \lg n)$. إلا أن المعاملات الثابتة المخبأة في تدوين_Θ مختلفة. ومع أن الفرز حسب الأساس يمكن أن يمرَّ (على n مفتاحًا) عددًا من المرّات أقلَّ من الفرز السريع، فإن كلَّ مرورٍ للفرز حسب الأساس يمكن أن يستغرق زمنًا أطول بكثير من نظيره في الفرز السريع. لذا فإن اختيارنا لخوارزمية الفرز الفضلى يعتمد على مميزات التنجيز، وعلى الحاسوب المستخدم (على سبيل المثال، غالبًا ما يستخدم الفرز السريع عتاديات الذاكرة السريعة بفعالية أكبر من الفرز

حسب الأساس)، وعلى معطيات الدخل. يضاف إلى ذلك، أن نسخة الفرز حسب الأساس - التي تَستخدم الفرز بالعد باعتباره فرزًا مستقرًا وسيطًا - لا تَفرز المعطيات في المكان in-place، على حين أن كثيرًا من طرق الفرز بالمقارنة التي تستغرق زمنًا $\Theta(n \lg n)$ تَفرز المعطيات في المكان. وبناء على ذلك، إذا احتلت ذاكرةُ التخزين الأولية المقامَ الأول في الأهمية، يمكننا تفضيل خوارزمية فرزٍ تُنَفَّذ في المكان كالفرز السريع مثلاً.

تمارين

1-3.8

استخدم الشكل 3.8 نموذجًا، واشرحْ عَمَل RADIX-SORT على لائحة الكلمات الإنكليزية التالية: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

2-3.8

أيٌّ من خوارزميات الفرز التالية مستقرة: الفرز بالإدراج، الفرز بالدمج، الفرز بالكومة، الفرز السريع؟ أعطِ آليةً scheme بسيطة تجعل أية خوارزمية فرز مستقرة. كم من الزمن الإضافي والذاكرة تستلزم آليتك؟

3-3.8

استخدم الاستقراء لتبرهن أن الفرز حسب الأساس يعمل كما يجب. أين يتطلب برهانك افتراض كون الفرز الوسيط مستقرًّا؟

4-3.8

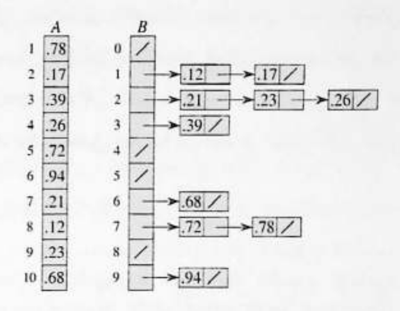
بيِّنْ كيف نفرز n عددًا صحيحًا في المجال من 0 إلى $n^3 - 1$ في زمن $O(n)$.

★ *5-3.8*

ما هو بالضبط عدد مرات الفرز اللازمة، في أول خوارزمية لفرز البطاقات وردت في هذا المقطع، وذلك لفرز أعداد عشرية من d خانة؟ وما هو عدد كدسات البطاقات التي قد يحتاج إليها العامل ليتابع البطاقات على نحو مستمر في أسوأ الحالات؟

4.8 الفرز بالدلاء

يفترض ***الفرز بالدلاء bucket sort*** أن الدخلَ مستمَدٌّ من توزيعٍ منتظم uniform distribution، وأن زمن تنفيذ هذا الفرز في الحالة الوسطى $O(n)$. إن الفرز بالدلاء سريع مثل الفرز بالعد، لأنه يفترض بعض الفرضيات على الدخل. ففي حين يفترض الفرز بالعد أن الدخل يتكون من أعداد صحيحة من مجال صغير، يفترض الفرز بالدلاء أن الدخل مولَّدٌ من إجرائيةٍ عشوائيةٍ تُوزِّع العناصرَ توزيعًا منتظمًا ومستقلاً في المجال $[0,1)$. (انظر المقطع ت.2، ففيه تعريف التوزيع المنتظم.)



الشكل 4.8 تطبيق BUCKET-SORT في حال $n = 10$. (أ) صفيفة الدخل $A[1..10]$. (**ب**) الصفيفة $B[0..9]$ من لوائح مفروزة (دلاء) بعد السطر 8 للخوارزمية. يضم الدلو i القيم التي تقع في المجال نصف-المفتوح $[i/10, (i+1)/10)$. يتكون الخرج المفروز من ضَم اللوائح $B[0], B[1], \ldots, B[9]$ ، الترتيب.

يَقْسم الفرزُ بالدلاء المجالَ $[0,1)$ إلى n مجالاً جزئيًا متساويًا أو ***دلاء buckets***، ثم يوزِّع الدخل المتمثل في n عددًا في هذه الدلاء. ولما كان الدخل موزَّعًا توزيعًا منتظمًا ومستقلاً في المجال $[0,1)$، فإننا لا نتوقع وقوع عدد كبير من الأعداد في كل دلو. وللحصول على الخرج، ما علينا إلا أن نفرز الأعدادَ في كل دلو، ثم نَمُرّ على الدلاء بالترتيب، ونضع عناصر كلٍّ منها في لائحة.

يفترض رمازنا للفرز بالدلاء أن الدخل هو صفيفة A من n عنصرًا، وأن كلَّ عنصرٍ $A[i]$ في الصفيفة يحقق المتراجحة $0 \le A[i] < 1$. يحتاج الرماز إلى صفيفة مساعدة $B[0..n-1]$ من لوائح مُترابطة (دلاء) ويفترض وجود آلية للمحافظة على هذه اللوائح. (يصف المقطع 2.10 كيفية تنجيز العمليات الأساسية على اللوائح المترابطة.)

```
BUCKET-SORT(A)
1  n = A.length
2  let B[0..n − 1] be a new array
3  for i = 0 to n − 1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊n A[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

يبين الشكل 4.8 عملية الفرز بالدلاء على صفيفة دخل من 10 أعداد.

حتى نتأكد من صحة عمل هذه الخوارزمية، نتأمّل العنصرين $A[i]$ و $A[j]$. نفترض، دون أن يؤثر ذلك

على الحالة العامة، أن $A[i] \le A[j]$. لما كان $\lfloor n\,A[i] \rfloor \le \lfloor n\,A[j] \rfloor$، فإن العنصر $A[i]$ سيدخل إما ضمن الدلو نفسه مع $A[j]$ أو في دلو دليله أقل. فإذا دخل $A[i]$ و $A[j]$ في الدلو نفسه، فإن حلقة **for** في السطرين 7-8 تضعهما في الترتيب الصحيح. وإذا دخلت $A[i]$ و $A[j]$ في دلوَيْن مختلفين، فإن السطر 9 يضعهما في الترتيب الصحيح. لذلك، فإن الفرز بالدلاء يعمل على نحو صحيح.

لتحليل زمن التنفيذ، لاحظ أن جميع الأسطر ما عدا السطر 8 يستغرق تنفيذها زمنًا $O(n)$ في أسوأ الحالات. نحتاج إلى تحليل الزمن الكلي اللازم لاستدعاء فرز بالإدراج n مرةً في السطر 8.

ولتحليل كلفة استدعاءات الفرز بالإدراج، نفترض أن n_i المتحول العشوائي الذي يشير إلى عدد العناصر الموضوعة في $B[i]$. ولما كان الفرز بالإدراج يُنفَّذ في زمنٍ تربيعيّ (انظر المقطع 2.2)، فإن زمن تنفيذ الفرز بالدلاء يساوي

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) .$$

ندرس الآن زمن تنفيذ الفرز بالدلاء في الحالة الوسطى، وذلك بحساب قيمة التوقع لزمن التنفيذ، حيث نحسب التوقع اعتمادًا على التوزيع الاحتمالي للدخل. بأخذ توقع الطرفين وباستخدام خطية التوقع، نحصل على

$$\begin{aligned}
\mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\
&= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}\left[O(n_i^2)\right] && \text{(اعتمادًا على خطية التوقع)} \\
&= \Theta(n) + \sum_{i=0}^{n-1} O(\mathrm{E}[n_i^2]) && \text{((اعتمادًا على المعادلة (ت.22))} \qquad (1.8)
\end{aligned}$$

ندّعي أنّ

$$\mathrm{E}\left[n_i^2\right] = 2 - 1/n \qquad (2.8)$$

لقيم $i = 0, 1, \ldots, n-1$. ليس من المفاجئ أن يكون لكل دلو i القيمة نفسها لـ $\mathrm{E}[n_i^2]$، لأن كل قيمة في صفيفة الدخل A تقع باحتمال متساوٍ في أي دلو. ولبرهان المعادلة (2.8) نُعرِّف المتحولات العشوائية المؤشرة (انظر المقطع 2.5)

$$X_{ij} = \mathrm{I}\,\{A[i] \text{ falls in bucket } i\}$$

حيث $i = 0, 1, \ldots, n-1$ و $j = 1, 2, \ldots, n$. ومنه،

$$n_i = \sum_{j=1}^{n} X_{ij}$$

ولحساب $\mathrm{E}[n_i^2]$، فإننا ننشر التربيع ونعيد تجميع الحدود:

$$\begin{aligned}
\mathrm{E}[n_i^2] &= \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} X_{ij}X_{ik}\right] \\
&= \sum_{j=1}^{n} \mathrm{E}[X_{ij}^2] + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} \mathrm{E}[X_{ij}X_{ik}] , \qquad (3.8)
\end{aligned}$$

حيث يَنتج السطر الأخير من خطية التوقع. سوف نحسب كلَّ مجموعٍ على حدة. إن المتحول العشوائي المؤشر X_{ij} يساوي 1 باحتمال $1/n$ و 0 ما عدا ذلك، ولذا يكون لدينا

$$\begin{aligned}
\mathrm{E}[X_{ij}^2] &= 1^2 \times \frac{1}{n} + 0^2 \times \left(1 - \frac{1}{n}\right) \\
&= \frac{1}{n} .
\end{aligned}$$

عندما $k \ne j$، فإن المتحولَيْن X_{ij} و X_{ik} مستقلان، ومنه

$$\begin{aligned}
\mathrm{E}[X_{ij}X_{ik}] &= \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}] \\
&= \frac{1}{n} \times \frac{1}{n} \\
&= \frac{1}{n^2} .
\end{aligned}$$

بتعويض هاتين القيمتَيْن المتوقعتين في المعادلة (3.8)، نحصل على

$$\begin{aligned}
\mathrm{E}[n_i^2] &= \sum_{j=1}^{n} \frac{1}{n} + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} \frac{1}{n^2} \\
&= n \times \frac{1}{n} + n(n-1) \times \frac{1}{n^2} \\
&= 1 + \frac{n-1}{n} \\
&= 2 - \frac{1}{n} ,
\end{aligned}$$

وهذا ما يبرهن المعادلة (2.8).

باستخدام قيمة التوقع في المعادلة (1.8)، نستنتج أن زمن تنفيذ الفرز بالدلاء في الحالة الوسطى يساوي $\Theta(n) + n \times O(2 - 1/n)) = \Theta(n)$.

إن الفرز بالدلاء لايزال ممكن التنفيذ في زمن خطي، ولو كان الدخل غير ناتج عن توزيع منتظم. فمادام الدخلُ يتمتع بخاصية كون مجموعِ مربعاتِ حجوم الدلاء خطيًّا مع العدد الكلي للعناصر، فإن المعادلة (1.8) تبيِّن أن الفرز بالدلاء سوف ينفَّذ في زمن خطيّ.

تمارين

1-4.8

استخدم الشكل 4.8 نموذجًا، لشرح عَمَل BUCKET-SORT على الصفيفة

$A = \langle .79,\ .13,\ .16,\ .64,\ .39,\ .20,\ .89,\ .53,\ .71,\ .42 \rangle$

2-4.8

علِّل لماذا يكون زمن تنفيذ خوارزمية الفرز بالدلاء في أسوأ الحالات $\Theta(n^2)$؟ كيف يمكن أن نجري تعديلاً بسيطًا على الخوارزمية بحيث يحافظ على زمن التنفيذ في الحالة الوسطى ويجعل زمن التنفيذ في أسوأ الحالات $O(n \lg n)$؟

3-4.8

ليكن X متحولاً عشوائيًّا يساوي عدد مرات الحصول على وجه heads في رميتين لقطعة نقود عادلة. ما هي قيمة $\mathrm{E}[X^2]$؟ وما هي قيمة $\mathrm{E}^2[X]$؟

4-4.8

ليكن لدينا n نقطة ضمن الدائرة الواحدية $p_i = (x_i, y_i)$، بحيث يكون $0 < x_i^2 + y_i^2 \leq 1$ لكل $i = 1, 2, \ldots, n$. افترض أن النقاط موزعة بانتظام؛ أي إن احتمال وجود نقطة في أية منطقة من الدائرة يتناسب مع مساحة تلك المنطقة. صمِّم خوارزميةً يكون زمن تنفيذها في الحالة الوسطى $\Theta(n)$ لفرز n نقطةً تبعًا لبعد كل منها $d_i = \sqrt{x_i^2 + y_i^2}$ عن المبدأ. (*تلميح*: صممّ أحجام الدلاء في BUCKET-SORT كي يَظهر التوزيعُ المنتظم للنقاط في الدائرة الواحدية.)

★ *5-4.8*

تُعرَّف ***دالة التوزيع الاحتمالي probability distribution function*** $P(x)$ لمتحولٍ عشوائي X بالعلاقة $P(x) = \Pr\{X \leq x\}$. افترض أننا سحبنا لائحةً من n متحولاً عشوائيًّا $X_1, X_2, \ldots, X_n$ تتبع دالة توزيع احتمالي مستمر P قابل للحساب في زمن $O(1)$. بيّنْ كيف نفرز هذه الأعداد في زمن توقعه خطي في الحالة الوسطى.

مسائل

1-8 حدود احتمالية دنيا على الفرز بالمقارنة

نبرهن في هذه المسألة، أن $\Omega(n \lg n)$ يمثّل حدًّا أدنى احتماليًّا لزمنِ تنفيذِ أيِّ فرزٍ بالمقارنة حتميّ deterministic أو ذي عشوائيةٍ مضافة randomized على n عنصرَ دخلٍ متمايزًا. نبدأ بدراسةِ فرزٍ بالمقارنة حتميٍّ A مع شجرة قرار T_A. ونفترض أن كلَّ تبديلٍ لمُدخلات A له الاحتمال نفسه.

أ. افترض أن كلَّ ورقةٍ من T_A أُلصِقَ بها احتمال بلوغها دخلاً عشوائيًّا ما. برهن أن $n!$ ورقة تمامًا سوف يُلصَق بها $1/n!$ ويُلصق بالباقي 0.

ب. نرمز بـ $D(T)$ إلى طول المسار الخارجي لشجرة قرار T؛ أي إن $D(T)$ يساوي مجموع أعماق جميع أوراق T. ولتكن T شجرةَ قرارٍ عددُ أوراقها $k > 1$، ولتكن LT و RT الشجرتين الفرعيتين اليسرى واليمنى لـ T. بيّنْ أن $D(T) = D(LT) + D(RT) + k$.

ت. ليكن $d(k)$ القيمة الصغرى لـ $D(T)$ على جميع أشجار القرار T التي عدد أوراقها $k > 1$. بيّنْ أن $d(k) = \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$. (*تلميح*: فكِّر في شجرة قرار T عدد أوراقها k تحقق القيمة الصغرى. وافترض أن i_0 عدد الأوراق في LT و $k - i_0$ عدد الأوراق في RT.)

ث. برهن أن الدالةَ $i \lg i + (k-i)\lg(k-i)$ أصغريةٌ عند $i = k/2$، حيث $k > 1$ و i في المجال $1 \le i \le k-1$. استنتج أن $d(k) = \Omega(k \lg k)$.

ج. برهن أن $D(T_A) = \Omega(n! \lg(n!))$، واستنتج أن زمن التنفيذ في الحالة الوسطى لفرز n عنصرًا هو $\Omega(n \lg n)$.

لنناقش، الآن، خوارزمية فرز بالمقارنة ذات عشوائية مضافة B. يمكننا توسيع نموذج شجرة القرار ليتعامل مع العشوائية المضافة، وذلك بتعريف نوعين من العقد: عقد مقارنة اعتيادية وعقد "إضافة عشوائية randomization". تُنَمذِج عُقَدُ الإضافةِ العشوائيةِ الخيارَ العشوائيَّ من الشكل $\text{RANDOM}(1, r)$ الذي تستخدمه الخوارزمية B؛ حيث يكون للعقدة r ابنًا، يُمكن اختيار أيٍّ منهم باحتمال متساوٍ خلال تنفيذ الخوارزمية.

ح. بيّنْ أنه في أيِّ فرزٍ بالمقارنة ذي عشوائية مضافة B، يوجد فرزٌ حتميٌّ بالمقارنة A لا يتجاوز عددُ المقارنات المتوقع مثيلَه في الفرز B.

2-8 الفرز في المكان في زمن خطي

افترض أننا نريد فرز صفيفة من n تسجيلة معطيات، وأن مفتاح كل تسجيلة له القيمة 0 أو 1. لعلّ خوارزمية

فرز مثل هذه التسجيلات تمتلك بعضًا من المميزات الثلاث التالية المرغوب فيها:

1. تُنفذ الخوارزمية في زمن $O(n)$.
2. الخوارزمية مستقرة.
3. تفرز الخوارزمية في المكان، مستخدمة قدرًا ثابتًا وليس أكثر من مساحة التخزين إضافة إلى الصفيفة الأصلية.

أ. أعطِ خوارزمية تحقق المعيارين 1 و 2.

ب. أعطِ خوارزمية تحقق المعيارين 1 و 3.

ت. أعطِ خوارزمية تحقق المعيارين 2 و 3.

ث. هل يمكنك استخدام أي من خوارزميات الفرز التي اقترحتها في (أ) إلى (ت) طريقةً للفرز المُستخدم في السطر 2 في RADIX-SORT، بحيث تفرز RADIX-SORT n تسجيلةً تتكون مفاتيحها من b بتًّا في زمن $O(b\,n)$ ؟ اشرح كيف أو لِمَ لا.

ج. افترض أن مفاتيح n تسجيلة تقع في المجال من 1 إلى k. بيِّنْ كيف يمكن تعديل الفرز بالعد بحيث تُفرز التسجيلات في المكان في زمن $O(n+k)$. يمكنك استخدام مساحة تخزين $O(k)$ إضافة إلى صفيفة الدخل. هل خوارزميتك مستقرة؟ (*تلميح*: ما الذي كنت ستفعله في حالة $k = 3$؟)

8-3 *فرز عناصر ذات أطوال مختلفة*

أ. ليكن لدينا صفيفة من أعداد صحيحة، حيث يمكن أن يكون للأعداد الصحيحة المختلفة عددٌ مختلفٌ من الخانات، إلا أن عدد الخانات الكلي *لكل* الأعداد الصحيحة في الصفيفة يساوي n. بيِّنْ كيف يمكن فرز الصفيفة في زمن $O(n)$.

ب. ليكن لديك صفيفة من متتاليات محارف، حيث يمكن أن يكون لمتتاليات المحارف المختلفة عددٌ مختلفٌ من المحارف، إلا أن عدد المحارف الكلي في جميع متتاليات المحارف يساوي n. بيِّنْ كيف يمكن فرز متتاليات المحارف في زمن $O(n)$.

(انتبه إلى أن الترتيب المرغوب فيه هنا هو الترتيب الأبجدي المتعارف؛ a < ab < b، مثلاً.)

8-4 *أباريق الماء*

افترض أن لديك n إبريقًا أحمر و n إبريقًا أزرق، جميعها مختلفة في الشكل والحجم. تتسع الأباريق الحمر كميات مختلفة من الماء، كما هو حال الأباريق الزرقاء. إضافةً إلى ذلك، يوجد لكلِّ إبريقٍ أحمر إبريقٌ أزرق

يتسع كمية الماء نفسها، والعكس بالعكس.

تكمن مهمتك في تجميع الأباريق في أزواج من الأباريق الحمراء والزرقاء المتساوية السعة. لتحقيق ذلك، يمكنك القيام بالعملية التالية: اخترْ زوجًا من الأباريق أحدهما أحمر والآخر أزرق. املأ الإبريق الأحمر بالماء، ثم أفرغ هذا الماء في الإبريق الأزرق. ستعلم من هذه العملية أيًّا من الإبريقين الأحمر أم الأزرق يمكن أن يتسع إلى كمية مياه أكبر، أو إن كان لهما الحجم نفسه. افترض أن عملية المقارنة هذه تستغرق وحدة زمنية واحدة. تَكمن مهمتك في إيجاد خوارزمية تنفذ عددًا أصغريًّا من المقارنات لتحديد المجموعات. تذكَّر أنك لا تستطيع مقارنة إبريقَيْن أحمرَيْن أو إبريقَيْن أزرقَيْن مباشرة.

أ. صِفْ خوارزميةً حتمية تستخدم $\Theta(n^2)$ عملية مقارنة لتجميع الأباريق في أزواج.

ب. برهنْ أن الحد الأدنى لعدد المقارنات التي يجب أن تُنفذها الخوارزمية لحل هذه المسألة هو $\Omega(n \lg n)$.

ت. أعطِ خوارزميةً ذات عشوائية مضافة يكون عدد المقارنات المتوقع فيها هو $O(n \lg n)$، وبرهن أن هذا الحد صحيح. ما هو عدد المقارنات في أسوأ الحالات في خوارزميتك؟

5-8 *الفرز الوسطي*

افترض أننا، عوضًا عن فرز صفيفة، نريد فقط أن تتزايد العناصر في المتوسط. وبعبارةٍ أدق، نقول عن صفيفة A من n عنصرًا إنها ***k*–مرتبة *k-sorted*** إذا كانت العلاقة التالية محققة لكل قيم $i = 1, 2, \dots, n-k$

$$\frac{\sum_{j=i}^{i+k-1} A[j]}{k} \leq \frac{\sum_{j=i+1}^{i+k} A[j]}{k} .$$

أ. ماذا يعني أن تكون الصفيفة 1–مرتبة.

ب. أعطِ تبديلاً للأعداد $1, 2, \dots, 10$ بحيث تكون 2–مرتبة، دون أن تكون مرتبة.

ت. برهن أن صفيفة من n عنصرًا تكون k–مرتبة إذا وفقط إذا كان $A[i] \leq A[i+k]$ لكل قيم $i = 1, 2, \dots, n-k$.

ث. أعطِ خوارزمية تجعل صفيفة من n عنصرًا k–مرتبة في زمن $O(n \lg(n/k))$.

يمكننا أيضًا إيجاد حدٍّ أدنى للزمن اللازم لإنتاج صفيفة k–مرتبة، عندما تكون k ثابتة.

ج. بيِّنْ أنه يمكننا فرز صفيفة k–مرتبة طولها n في زمن $O(n \lg k)$. (*تلميح*: استخدم حل التمرين 5.6-9.)

ح. بيِّنْ أنه عندما تكون k ثابتة، نحتاج إلى زمن $\Omega(n \lg n)$ لجعل صفيفة من n عنصرًا k–مرتبة. (*تلميح*: استخدم حل الجزء السابق مع الحد الأدنى للفرز بالمقارنة.)

6-8 الحد الأدنى لدمج لوائح مرتبة

تَبْرُز مسألةُ دمج لائحتَيْن مرتبتين كثيرًا. وقد رأينا في المقطع 2.3-1 إجرائيةً لدمج لائحتَيْن مرتبتين بوصفها مساقًا فرعيًا يُدعى MERGE. سنُبرهن في هذه المسألة أن حدًّا أدنى يساوي $2n-1$ لعدد المقارنات اللازمة في أسوأ الحالات لدمج لائحتَيْن مرتبتين، تضم كل منهما n عنصرًا.

سنبيِّن أولاً، حدًّا أدنى للمقارنات $2n-o(n)$ باستخدام شجرة قرار.

أ. ليكن لدينا $2n$ عددًا، احسب عدد الطرق الممكنة لتقسيم العناصر إلى لائحتَيْن مرتبتين، تضم كل منهما n عنصرًا.

ب. بيِّنْ، باستخدام شجرة قرار وجوابك عن السؤال (أ)، أن أية خوارزمية تدمج لائحتَيْن مرتبتين دمجًا صحيحًا يجب أن تُنجز على الأقل $2n-o(n)$ مقارنة.

نبين الآن حدًّا أكثر ملاصقة نوعًا ما وهو $2n-1$.

ت. بيِّنْ أنه إذا كان عنصران متتاليان في ترتيب مفروز هما من لائحتَيْن مختلفتين، فلا بد أنه قد جرت مقارنتهما.

ث. استخدم إجابتك عن الجزء السابق لتبيِّن أن $2n-1$ هو حدٌّ أدنى للمقارنات لدمج لائحتَيْن مرتبتين.

7-8 توطئة الفرز 0-1 وخوارزمية columnsort

تأخذ عملية ***قارن–بادل*** ***compare-exchange*** على عُنصرَي صفيفة $A[i]$ و $A[j]$، حيث $i<j$، الصيغة التالية:

```
COMPARE-EXCHANGE(A, i, j)
1  if A[i] > A[j]
2      exchange A[i] with A[j]
```

نعلم، بعد تنفيذ العملية قارن–بادل، أن $A[i] \leq A[j]$.

تعمل ***خوارزمية قارن–بادل مُغفلة*** ***oblivious compare-exchange algorithm*** فقط على متتالية مُحددة سلفًا من عمليات قارن–بادل. يجب أن تكون أدلة المواقع المُراد مقارنتها في المتتالية محددة سلفًا، ومع أن هذه الأدلة قد تعتمد على عدد العناصر المفروزة، إلا أنه لا يمكنها أن تعتمد على القيم المفروزة، ولا على نتيجة أية عملية قارن–بادل سابقة. على سبيل المثال، نُعبِّر هنا عن فرز بالإدراج باعتباره خوارزمية قارن–بادل مغفلة:

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      for i = j - 1 downto 1
3          COMPARE-EXCHANGE(A, i, i + 1)
```

تقدم ***توطئة الفرز 0-1 0-1 sorting lemma*** طريقة فعالة للبرهان على أن خوارزمية قارن–بادل مغفلة تُقدم نتيجة مفروزة. تنص هذه التوطئة على أنه إذا كانت خوارزمية قارن–بادل المغفلة تفرز فرزًا صحيحًا كلَّ متتالية دخل مكونة من أصفارٍ ووحدانٍ فقط، فإنها تفرز فرزًا صحيحًا كل المُدخلات مهما تكن قيمها.

المطلوب برهان توطئة الفرز 0-1 بطريقة البرهان المعاكس الموجب contrapositive: إذا أخفقت خوارزمية قارن–بادل مغفلة في فرز دخل اعتباطي، فستُخفق في فرز دخلٍ ما من الأصفار والوحدان. افترض أن خوارزمية قارن–بادل مغفلة أخفقت في فرز الصفيفة $A[1..n]$ فرزًا صحيحًا. لتكن $A[p]$ أصغر قيمة في A تضعها الخوارزمية X في المكان الخاطئ، ولتكن $A[q]$ القيمة التي تضعها الخوارزمية X في المكان الذي من المفترض أن تذهب إليه $A[p]$. عرِّف صفيفة $B[1..n]$ من الأصفار والوحدان كما يلي:

$$B[i] = \begin{cases} 0 & \text{if } A[i] \le A[p] \,, \\ 1 & \text{if } A[i] > A[p] \,. \end{cases}$$

أ. ناقش أنه إذا كان $A[q] > A[p]$، فإن $B[p] = 0$ و $B[q] = 1$.

ب. لإتمام برهان توطئة الفرز 0-1، برهن أن الخوارزمية X تُخفق في فرز الصفيفة B فرزًا صحيحًا.

سوف تستخدم الآن توطئة الفرز 0-1 للبرهان على أن خوارزمية فرزٍ معينة تعمل بوجهٍ صحيح. تعمل الخوارزمية ***columnsort*** على صفيفة مستطيلة من n عنصرًا. تتكون الصفيفة من r سطرًا و s عمودًا (أي $n = rs$)، وتخضع لثلاثة قيود:

- يجب أن يكون r زوجيًّا،
- يجب أن تقبل r القسمة على s،
- $r \ge 2s^2$.

عند انتهاء columnsort، تكون الصفيفة ***مفروزة وفق أعمدتها column-major order***: أي إذا قرأنا الأعمدة نزولاً، من اليسار إلى اليمين، تكون العناصر متزايدة باطراد.

تتبع columnsort ثماني خطوات، بقطع النظر عن قيمة n. الخطوات الفردية كلها متماثلة: افرز كل عمود على حدة. تنجز كل خطوة زوجية تبديلاً محددًا. وهذه الخطوات هي:

1. افرز كل عمود.
2. انقل transpose الصفيفة، ولكن مع إعادة تشكيلها لتصبح r سطرًا و s عمودًا. بعبارة أخرى، ضع العمود الأيسر الأول في الأسطر r/s العليا، بالترتيب؛ ثم ضع العمود التالي في الأسطر r/s التالية، بالترتيب، وهكذا ...
3. افرز كل عمود.

(أ)

10	14	5
8	7	17
12	1	6
16	9	11
4	15	2
18	3	13

(ب)

4	1	2
8	3	5
10	7	6
12	9	11
16	14	13
18	15	17

(ت)

4	8	10
12	16	18
1	3	7
9	14	15
2	5	6
11	13	17

(ث)

1	3	6
2	5	7
4	8	10
9	13	15
11	14	17
12	16	18

(ج)

1	4	11
3	8	14
6	10	17
2	9	12
5	13	16
7	15	18

(ح)

1	4	11
2	8	12
3	9	14
5	10	16
6	13	17
7	15	18

(خ)

	5	10	16
	6	13	17
	7	15	18
1	4	11	
2	8	12	
3	9	14	

(د)

	4	10	16
	5	11	17
	6	12	18
1	7	13	
2	8	14	
3	9	15	

(ذ)

1	7	13
2	8	14
3	9	15
4	10	16
5	11	17
6	12	18

الشكل 5-8 خطوات columnsort. (أ) صفيفة الدخل من 6 أسطر و 3 أعمدة. **(ب)** بعد فرز كل عمود في الخطوة 1. **(ت)** بعد عملية النقل وإعادة التشكيل في الخطوة 2. **(ث)** بعد فرز كل عمود في الخطوة 3. **(ج)** بعد تنفيذ الخطوة 4، التي تنجز عكس التبديل المنفَّذ في الخطوة 2. **(ح)** بعد فرز كل عمود في الخطوة 5. **(خ)** بعد إزاحة بمقدار نصف عمود في الخطوة 6. **(د)** بعد فرز كل عمود في الخطوة 7. **(ذ)** بعد تنفيذ الخطوة 8، التي تنجز عكس التبديل المنفَّذ في الخطوة 6. الصفيفة الآن مفروزة وفق أعمدتها.

4. أنجز عكس التبديل المُنفَّذ في الخطوة 2.
5. افرز كل عمود.
6. انقل النصف الأعلى من كل عمود إلى النصف الأسفل من العمود نفسه، وانقل النصف الأسفل من كل عمود إلى النصف الأعلى من العمود الذي يليه يمينًا. اترك النصف الأعلى من العمود الأيسر الأول فارغًا. انقل النصف الأسفل من العمود الأيمن الأخير إلى النصف الأعلى من عمود أيمن أخير جديد تنشئه، واترك النصف الأسفل من هذا العمود الجديد فارغًا.
7. افرز كل عمود.
8. أنجز عكس التبديل المُنفَّذ في الخطوة 6.

يبين الشكل 5.8 مثالاً على خطوات columnsort في حالة $r = 6$ و $s = 3$. (ومع أن هذا المثال لا يحقِّق المطلب $r \geq 2s^2$، فإنه يعمل.)

ت. بيِّن أنه يمكننا معاملة columnsort كخوارزمية قارن-بادل مغفلة، ولو كنا لا نعلم طريقة الفرز التي تستخدمها الخطوات الفردية.

وعلى الرغم من أنه قد يبدو من الصعب التصديق أن columnsort تفرز حقًا، فإنك ستستخدم توطئة

الفرز 0-1 للبرهان على قيامها بذلك. من الممكن تطبيق توطئة الفرز 0-1 لكوننا نستطيع معاملة columnsort كخوارزمية قارن-بادل مغفلة. سوف يساعدك التعريفان التاليان في تطبيق توطئة الفرز 0-1. نقول عن منطقة في صفيفة أنها ***نظيفة clean*** إذا كنا نعلم أنها تحتوي إما أصفارًا فقط وإما وحدانًا فقط. فإذا كانت المنطقة تحتوي مزيجًا من الأصفار والوحدان، فهي ***ملوثة dirty***. افترض، من الآن فصاعدًا، أن الصفيفة تحتوي أصفارًا ووحدانًا فقط، وأنه يمكننا معاملتها كصفيفة من r سطرًا و s عمودًا.

ث. برهن أن الصفيفة، بعد الخطوات 1-3، تضم بعض الأسطر النظيفة في أعلاها، وبعض الأسطر النظيفة من الوحدان في أسفلها، و s سطرًا ملوثًا بينهما على الأكثر.

ج. برهن أنه إذا قرأنا أعمدة الصفيفة واحدًا تلو الآخر، بعد الخطوة 4، فإنها تبدأ بمنطقة نظيفة من الأصفار، وتنتهي بمنطقة نظيفة من الوحدان، وتمتلك منطقة ملوثة لا تتجاوز s^2 عنصرًا في الوسط.

ح. برهن أن الخطوات 5-8 تنتج خرجًا مفروزًا كليًا من الأصفار والوحدان. استنتج أن columnsort تفرز فرزًا صحيحًا كل المدخلات مهما تكن قيمها.

د. افترض الآن أن r لا يقبل القسمة على s. برهن أنه بعد الخطوات 1-3، تحتوي الصفيفة على بعض الأسطر النظيفة من الأصفار في القمة، وبعض الأسطر النظيفة من الوحدان في الأسفل، وعلى الأكثر $2s-1$ سطرًا ملوثًا بينهما. ما هو مقدار كِبَر r بالنسبة إلى s اللازم حتى تفرز columnsort فرزًا صحيحًا عندما لا تقبل r القسمة على s؟

ذ. اقترح تغييرًا بسيطًا للخطوة 1 بحيث تسمح لنا بالمحافظة على المطلب $r \geq 2s^2$ حتى عندما لا تقبل r القسمة على s، برهن على أن columnsort تفرز فرزًا صحيحًا، بوجود تعديلك.

ملاحظات الفصل

كان Ford و Johnson [110] أول من استخدم نموذج شجرة القرار لدراسة الفرز بالمقارنة. يتناول البحث الشامل لـ Knuth [211] في الفرز أشكالاً عديدة مختلفة لمسألة الفرز، ومنها الحد الأدنى النظري information-theoretic لتعقيد الفرز الذي تعرضنا له هنا. دَرَسَ Ben-Or [39] حدودًا دنيا للفرز باستخدام تعميمات لنموذج شجرة القرار.

يَنسِب Knuth إلى H. H. Seward اختراع الفرز بالعد في عام 1945، وكذلك فكرة الجمع بين الفرز بالعد والفرز حسب الأساس. يبدو أن الفرز حسب الأساس الذي يبدأ بالخانة الأقل قيمة كان خوارزمية شعبية استخدمها مشغلو فارزات البطاقات الميكانيكية على نطاق واسع. تبعًا لـ Knuth، فإن أول إشارة

منشورة لهذه الطريقة هي وثيقة لـ L. J. Comrie عام 1929 تصف معدات للبطاقات المثقبة. تُستخدم خوارزمية الفرز بالدلاء منذ عام 1956، عندما اقترح الفكرة الأساسية E. J. Isaac و R. C. Singleton [188].

قَدَّم Munro و Raman [263] خوارزمية فرز مستقرة تُنجز $O(n^{1+\epsilon})$ مقارنةً في أسوأ الحالات، حيث $0 < \epsilon \leq 1$ أي ثابت محدد. ومع أن كل الخوارزميات التي تُنَفَّذ في زمن $O(n \lg n)$ تقوم بعددٍ أقل من المقارنات، فإن خوارزمية Munro و Raman تُحرِّك المعطيات $O(n)$ مرةً فقط، وهي تعمل في المكان.

دَرَسَ العديد من الباحثين حالة فرز n عددًا صحيحًا من b بتًّا في زمن $o(n \lg n)$. وقد جرى الوصول إلى العديد من النتائج الإيجابية، كل منها تعتمد على فرضيات مختلفة قليلاً بالنسبة إلى نموذج الحوسبة والقيود المفروضة على الخوارزمية. تفترض كل النتائج أن ذاكرة الحاسوب مقسمة إلى كلمات من b بتًّا قابلة للعنونة. قَدَّم Fredman و Willard [115] بنية معطيات شجرة صهر fusion tree واستخدماها لفرز n عددًا صحيحًا في زمن $O(n \lg n / \lg \lg n)$. وحسَّنَ Andersson [16] هذا الحد لاحقًا إلى $O(n\sqrt{\lg n})$. تحتاج هذه الخوارزميات إلى استخدام عمليات جداء والعديد من الثوابت المحسوبة سلفًا. بيَّنَ كلٌّ من Andersson و Hagerup و Nilsson و Raman [17] كيف يمكن فرز n عددًا صحيحًا في زمن $O(n \lg \lg n)$ دون استخدام عمليات جداء، ولكن طريقتهم تتطلب ذاكرة تخزين يمكن أن تكون غير محدودة بدلالة n. يمكن، باستخدام تلبيد جدائي multiplicative hashing، إنقاص حجم ذاكرة التخزين اللازمة إلى $O(n)$، ولكن يصبح الحد $O(n \lg \lg n)$ في أسوأ حالات زمن التنفيذ الوسطي حدَّ الزمن المتوقع expected-time bound. قَدَّم Thorup [335] تعميمًا لأشجار البحث الأسية لـ Andersson [16]، بخوارزمية فرز في زمن $O(n(\lg \lg n)^2)$ لا تستخدم الجداء ولا عشوائية مضافة، وتَستخدم مساحة تخزين خطية. وحَسَّن Han [158] حدَّ الفرز إلى $O(n \lg \lg n \lg \lg \lg n)$، وذلك بمزج هذه التقنيات مع بعض الأفكار الجديدة. ومع أن هذه الخوارزميات تمثل قفزات نظرية ذات أهمية، لكنها جميعًا معقدة إلى حدٍّ ما، ومن المستبعد في الوقت الحالي أن تنافس خوارزميات الفرز الراهنة المستخدَمة.

تُنسَب خوارزمية columnsort في المسألة 8-7 إلى Leighton [227].

9 الأوساط وإحصائيات الترتيب

إن ***إحصائية الترتيب*** i (*i th **order statistic***) لمجموعة من n عنصرًا هي العنصر الأصغر ذو الترتيب i. على سبيل المثال، إن ***الأصغر minimum*** لمجموعة من العناصر هو إحصائية الترتيب الأولى ($i = 1$)، ***والأكبر maximum*** هو إحصائية الترتيب n ($i = n$). ***والوسط median***، على نحو غير صوري، هو النقطة في "الوسط" لمجموعة. فإذا كان n فرديًّا، فإن الوسط وحيد، ويقع عند $i = (n + 1)/2$. وإذا كان n زوجيًّا، فلدينا وسطان يقعان عند $i = n/2$ و $i = n/2 + 1$. وهكذا، وبقطع النظر عن ازدواجية (ندية parity) n، فإن الوسطَيْن يقعان عند $i = \lfloor(n + 1)/2\rfloor$ (***الوسط الأدنى lower median***) و $i = \lceil(n + 1)/2\rceil$ (***الوسط الأعلى upper median***). وللتبسيط، سنستخدم في هذا الكتاب، على الدوام، التعبير "الوسط the median" للإشارة إلى الوسط الأدنى.

يناقش هذا الفصل مسألة اختيار إحصائية الترتيب i لمجموعة من n عددًا متمايزًا. وسنفترض للملاءمة أن المجموعة تحتوي أعدادًا متمايزة، مع أننا فعليًّا يمكننا تعميم كل شيء نقوم به على الحالة التي تحتوي فيها المجموعة قيمًا مكررة. يمكننا تحديد ***مسألة الاختيار selection problem*** صوريًّا كما يلي:

الدخل: مجموعة A من n عددًا (متمايزًا) وعددٍ صحيحٍ i، حيث $1 \le i \le n$.

الخرج: العنصر $x \in A$ الذي هو أكبر تمامًا من $i - 1$ عنصرًا سواه في المجموعة A.

يمكننا حل مسألة الاختيار بزمن $O(n \lg n)$، حيث يمكننا فرز الأعداد باستخدام الفرز بالكومة أو بالدمج، ثم نشير ببساطة إلى العنصر ذي الدليل i في صفيفة الخرج. يعرض هذا الفصل خوارزميات أسرع.

نناقش، في المقطع 1.9، مسألة اختيار أصغر عنصر في مجموعة من العناصر وأكبر عنصر فيها. وتُعتَبر مسألة الاختيار العامة هي المسألة الأكثر أهمية التي سندرسها في المقطعين التاليين. يُحلِّل المقطع 2.9 خوارزمية ذات عشوائية مضافة تُحقق زمن تنفيذ متوقع $O(n)$، وذلك بافتراض أن العناصر متمايزة. ويتضمن المقطع 3.9 خوارزمية ذات أهمية نظرية كبيرة تُحقِّق زمن تنفيذ $O(n)$ في أسوأ الحالات.

1.9 الأصغر والأكبر

ما هو عدد المقارنات اللازمة لتحديد العنصر الأصغر في مجموعة من n عنصرًا؟ يمكننا بسهولة الحصول على حدٍّ أعلى لـ $n-1$ عملية مقارنة: نفحص كل عنصر من المجموعة بالترتيب (واحدًا تلو الآخر) ونحتفظ بأثر أصغر عنصر جرى العثور عليه آنفًا. سنفترض، في الإجراء التالي، أن المجموعة موجودة في صفيفةٍ A طولها $A.length = n$.

```
MINIMUM(A)
1  min = A[1]
2  for i = 2 to A.length
3      if min > A[i]
4          min = A[i]
5  return min
```

يمكننا بالطبع، أيضًا، إيجاد أكبر عنصر بإجراء $n-1$ مقارنة.

ولكن، هل هذا أقصى ما نستطيع فعله؟ الجواب: نعم، لأنه يمكننا الحصول على الحد الأدنى لـ $n-1$ مقارنةً لمسألةِ تحديدِ أصغرِ عنصر. تخيَّل أن أيَّ خوارزميةٍ تحدِّد أصغرَ عنصر هي مبارياتُ دَوْري بين العناصر، وأن كلَّ مقارنةٍ هي مباراةٌ في الدوري يربحها أصغر العنصرين. وبملاحظة أن كل عنصر ما عدا الرابح سيخسر على الأقل مباراة واحدة، يمكننا أن نستنتج أننا بحاجة إلى $n-1$ مقارنةً لتحديد العنصر الأصغر. وعلى ذلك فإن الخوارزمية MINIMUM أمثليةٌ فيما يتعلَّق بعدد المقارنات المنجزة.

إيجاد الأصغر والأكبر في آن واحد (معًا)

يتعيَّن علينا، في بعض التطبيقات، إيجاد العنصر الأصغر والعنصر الأكبر لمجموعةٍ من n عنصرًا في آنٍ معًا. فمثلاً، يمكن أن يتطلب برنامج بياني تقييس مجموعة من المعطيات (x, y) لتتوافق مع شاشة إظهار مستطيلة أو تجهيزة إظهار بيانية أخرى. ولتحقيق ذلك، يجب أن يوجد البرنامج أولاً القيمة الصغرى والكبرى لكلِّ إحداثيّ.

ينبغي، في هذه المرحلة، أن تتضح كيفية إيجاد الأصغر والأكبر معًا لـ n عنصرًا بإجراء $\Theta(n)$ مقارنة، وهي أمثليةٌ تقاربيًّا: نوجد الأصغر والأكبر بصورةٍ مستقلة باستخدام $n-1$ مقارنةً لكل منهما، أي $2n-2$ مقارنة لكليهما.

في الحقيقة، يمكننا إيجاد الأصغر والأكبر معًا بإجراء $3\lfloor n/2 \rfloor$ مقارنةً على الأكثر. يمكننا فعل ذلك بالاحتفاظ بالأصغر والأكبر اللذين عُثِر عليهما آنفًا. وعوضًا عن معالجة كلِّ عنصرٍ دخلٍ بمقارنته بالأصغر والأكبر الحاليين، بتكلفةِ مقارنتَيْن لكلِّ عنصر، نعالِج العناصر أزواجًا. وذلك بأن نبدأ بمقارنةِ زوجٍ من عناصر

الدخل *أحدهما بالآخر*، ثم نقارن أصغرهما بالأصغر الحالي وأكبرهما بالأكبر الحالي، وتكلفة هذا هو ثلاث مقارنات لكلّ عنصرين.

يعتمد تحديد القيمتَيْن الابتدائيتين للأصغر والأكبر الحاليين على كون n عددًا فرديًّا أو زوجيًّا. فإذا كان n فرديًّا، فإننا نجعل الأصغر والأكبر كليهما يساوي قيمةَ العنصر الأول، ثم نعالج العناصر المتبقية أزواجًا. وإذا كان n زوجيًّا، فإننا ننجز مقارنة بين العنصرَيْن الأولَيْن لتحديد القيمتَيْن الابتدائيتين للأصغر والأكبر، ثم نعالج العناصر المتبقية أزواجًا كما في حالة n الفردية.

لنحلِّل الآن العدد الكلي للمقارنات: إذا كان n فرديًّا، فإننا ننجز $3\lfloor n/2 \rfloor$ مقارنة. وإذا كان n زوجيًّا، فإننا ننجز عملية مقارنة أولية واحدة يتبعها $3(n-2)/2$ مقارنة، أي $3n/2-2$ مقارنة كلية. وهكذا، فإن العدد الكلي للمقارنات يساوي – في كلتا الحالتين – $3\lfloor n/2 \rfloor$ على الأكثر.

تمارين

1-1.9

بيّن أنه يمكننا إيجاد العنصر الثاني في الصغر لـ n عنصرًا بإجراء $n + \lceil \lg n \rceil - 2$ مقارنة في أسوأ الحالات. (*تلميح*: أوجد أيضًا العنصر الأصغر.)

★ *2-1.9*

برهن أن $\lceil 3n/2 \rceil - 2$ هو الحد الأدنى لعدد عمليات المقارنة اللازم لإيجاد الأصغر والأكبر معًا لـ n عنصرًا في أسوأ الحالات. (*تلميح*: ادرس عدد الأعداد التي يمكن أن تكون إما الأصغر وإما الأكبر، وتحرَّ كيف تؤثر المقارنة على هذه الأعداد.)

2.9 الاختيار بزمن خطي متوقع

تبدو مسألة الاختيار العامة أكثر صعوبة من المسألة البسيطة لإيجاد الأصغر. ومع ذلك، فإن من المدهش أن زمن التنفيذ المقارب لكلا المسألتين هو نفسه: وهو $\Theta(n)$. سنقدم في هذا المقطع خوارزمية فرِّق-تسد divide-and-conquer لحل مسألة الاختيار. نُمذجت الخوارزمية RANDOMIZED-SELECT على نمط خوارزمية الفرز السريع المشروحة في الفصل 7. وكما في الفرز السريع، فإننا نجزّئ صفيفة الدخل عَوْديًّا. ولكن على عكس الفرز السريع، الذي يعالج كلا طرفي التجزئة، فإن RANDOMIZED-SELECT يعمل على طرف واحد فقط من التجزئة. يتضح هذا الاختلاف في التحليل: ففي حين أن زمن التنفيذ المتوقع للفرز السريع هو $\Theta(n \lg n)$، فإن زمن التنفيذ المتوقع لـ RANDOMIZED-SELECT هو $\Theta(n)$، بافتراض أن العناصر متمايزة.

يَستخدم الإجراءُ RANDOMIZED-SELECT الإجراءَ RANDOMIZED-PARTITION المقدَّم في المقطع 3.7.

ومن ثَم، فإن هذا الإجراء، كما في RANDOMIZED-QUICKSORT، هو خوارزمية ذات عشوائية مضافة randomized algorithm، لكون سلوكها يتحدد جزئيًا بواسطة خرج مولِّد أعداد عشوائية. يعيد الرماز التالي لـ RANDOMIZED-SELECT العنصر الأصغر ذي الترتيب i للصفيفة $A[p\,..\,r]$.

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2      return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q - p + 1
5  if i == k          // the pivot value is the answer
6      return A[q]
7  elseif i < k
8      return RANDOMIZED-SELECT(A, p, q - 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```

يعمل الإجراء RANDOMIZED-PARTITION كما يلي. يفحص السطر 1 الحالة الأساسية للعَوْدية، التي تتكون فيها الصفيفة الجزئية $A[p\,..\,r]$ من عنصر واحد. في هذه الحالة، يجب أن تساوي i القيمة 1، ونعيد ببساطة $A[p]$ في السطر الثاني على أنه العنصر الأصغر ذو الترتيب i. وإلا فإن استدعاء RANDOMIZED-PARTITION في السطر الثالث من الخوارزمية، يجزّئ الصفيفة $A[p\,..\,r]$ إلى صفيفتين جزئيتين (يمكن أن تكونا خاليتين) $A[p\,..\,q-1]$ و $A[q+1\,..\,r]$ بحيث يكون كل عنصر من $A[p\,..\,q-1]$ أصغر أو يساوي $A[q]$، والذي بدوره أصغر من كل عنصر في $A[q+1\,..\,r]$. وكما في الفرز السريع، سوف نشير إلى $A[q]$ بالعنصر ***المحوري pivot***. يحسب السطر 4 عدد العناصر k في الصفيفة الجزئية $A[p\,..\,q]$، أي عدد العناصر في الجانب الأدنى من التجزئة زائد واحد هو العنصر المحوري. بعدها، يفحص السطر 5: هل $A[q]$ هو العنصر الأصغر ذو الترتيب i؟ فإذا كان الأمر كذلك، يعيد السطر 6 العنصر $A[q]$. وإلا فإن الخوارزميةَ تحدِّد في أيٍّ من الصفيفتَيْن الجزئيتين $A[p\,..\,q-1]$ و $A[q+1\,..\,r]$ يقع العنصر الأصغر ذو الترتيب i. فإذا كان $i < k$، فإن العنصر المنشود يقع في الجانب الأدنى من التجزئة، ويختاره السطر 8 عَوْديًا من الصفيفة الجزئية. وإذا كان $i > k$، فإن العنصر المنشود يقع في الجانب الأعلى من التجزئة. ولما كنا نعلم سلفًا وجود k قيمةً جميعها أصغر من العنصر الأصغر ذي الترتيب i لـ $A[p\,..\,r]$ (وهي بالتحديد عناصر الصفيفة $A[p\,..\,q]$) فإن العنصرَ المنشودَ هو العنصرُ الأصغرُ ذو الترتيب $(i-k)$ لـ $A[q+1\,..\,r]$، وهو الذي يعثر عليه السطر 9 عَوْديًا. يبدو أن الرماز يَسمح باستدعاءٍ عَوْديٍّ لصفيفة جزئية خالية العناصر. سيُطلب إليك في التمرين 9.2-1 أن تبيّن أن هذه الحالة غير ممكنة الحدوث.

إن زمن تنفيذ RANDOMIZED-SELECT في أسوأ الحالات هو $\Theta(n^2)$، حتى في حال إيجاد الأصغر، لأنه من الممكن أن نكون سيِّئي الحظ تمامًا ونجزّئ دائمًا حول أكبر العناصر المتبقية، وتستغرق التجزئة

زمنًا $\Theta(n)$. وسوف نرى أن للخوارزمية زمن تنفيذ متوقع خطي، لذلك ولكونها ذات عشوائية مضافة، فلا يوجد أي دخل خاص يظهر السلوك في أسوأ الحالات.

لتحليل زمن التنفيذ المتوقع لـ RANDOMIZED-SELECT، سنفترض أن زمن التنفيذ لصفيفة $A[p\,..\,r]$ من n عنصرًا هو متحول عشوائي نشير إليه بـ $T(n)$، ونحصل على الحد الأعلى لـ $\mathrm{E}[T(n)]$ كما يلي. يعيد الإجراء RANDOMIZED-PARTITION باحتمالات متساوية أيَّ عنصرٍ على أنه العنصر المحوري. لذلك، فإن الصفيفة $A[p\,..\,q]$ لها k عنصرًا (جميعها أصغر من العنصر المحوري أو تساويه) باحتمال $1/n$، وذلك لكلِّ k حيث $1 \le k \le n$. نُعرِّف للقيم $k = 1, 2, \ldots, n$، متحولات عشوائية مؤشِّرة indicator random variables X_k حيث:

$$X_k = \mathrm{I}\{\text{the subarray } A[p\,..\,q] \text{ has exactly } k \text{ elements}\} \ ,$$

ومنه، إذا افترضنا أن العناصر متمايزة، يكون لدينا

$$\mathrm{E}[X_k] = 1/n \ . \tag{1.9}$$

حين نستدعي RANDOMIZED-SELECT ونختار $A[q]$ باعتباره عنصرًا محوريًّا، فإننا لا نعلم سلفًا إذا كنا سننهي فورًا بالجواب الصحيح، أم سنستدعي الإجراء عَوْديًّا للصفيفة الجزئية $A[p\,..\,q-1]$، أم سنستدعي الإجراء عَوْديًّا للصفيفة الجزئية $A[q+1\,..\,r]$. يعتمد هذا القرار على مكان وقوع العنصر الأصغر ذي الترتيب i بالنسبة إلى $A[q]$. فإذا افترضنا أن $T(n)$ متزايد باطراد، أمكننا تقييدُ الزمن-الأعلى اللازم للاستدعاء العَوْدي بالزمن اللازم للاستدعاء العَوْدي لأكبر دخل ممكن. وبعبارة أخرى، للحصول على حدٍّ أعلى، سنفترض أن العنصر ذا الترتيب i يقع دائمًا في جانب التجزئة التي لها أكبر عدد من العناصر. إذا كان لدينا استدعاء معطى للإجراء RANDOMIZED-SELECT، فإن المتحول العشوائي المؤشِّر X_k يأخذ القيمة 1 لإحدى قيم k فقط، والقيمة 0 لقيم k الأخرى. فإذا كان $X_k = 1$، فإن حجم الصفيفتَيْن الجزئيتين (اللتين يمكن أن يستدعيهما الإجراء عَوْديًّا) هو: $k-1$ و $n-k$. ويكون لدينا التكرار:

$$\begin{aligned} T(n) &\le \sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big) \\ &= \sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n) \ . \end{aligned}$$

وبأخذ القيم المتوقعة للطرفين، يكون لدينا

$$\begin{aligned} \mathrm{E}[T(n)] &\le \mathrm{E}\left[\sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n)\right] \\ &= \sum_{k=1}^{n} \mathrm{E}[X_k \cdot T(\max(k-1, n-k))] + O(n) \qquad \text{(من خطية التوقع)} \end{aligned}$$

$$= \sum_{k=1}^{n} \mathrm{E}[X_k] \cdot \mathrm{E}[T(\max(k-1, n-k))] + O(n) \qquad \text{(من العلاقة (ت − 24))}$$

$$= \sum_{k=1}^{n} \frac{1}{n} \cdot \mathrm{E}[T(\max(k-1, n-k))] + O(n) \,. \qquad \text{(من العلاقة (1.9))}$$

ولتطبيق العلاقة (ت−24)، فإننا نُعَوِّل على كون X_k و $T(\max(k-1, n-k))$ متحولين عشوائيين مستقلين. يُطلب إليك في التمرين 9.2-2 تبرير هذا الادعاء.

ليكن لدينا التعبير $\max(k-1, n-k)$. لدينا

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lceil n/2 \rceil \,, \\ n-k & \text{if } k \le \lceil n/2 \rceil \,. \end{cases}$$

فإذا كان n زوجيًا، يَظهر كل حدٍّ من الحدود من $T(\lceil n/2 \rceil)$ إلى $T(n-1)$ مرتين تمامًا في المجموع، وإذا كان n فرديًا، تَظهر كل هذه الحدود مرتين ويَظهر الحد $T(\lfloor n/2 \rfloor)$ مرة واحدة. وبذلك يكون لدينا:

$$\mathrm{E}[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathrm{E}[T(k)] + O(n) \,.$$

يتبيَّن بالتعويض أن $\mathrm{E}[T(n)] = O(n)$. لنفترض أن $\mathrm{E}[T(n)] \le cn$ لثابتٍ c يحقق الشروط الابتدائية للعَوْدية. سنفترض أن $T(n) = O(1)$ لقيم n التي هي أقل من ثابتٍ ما (نحدِّده لاحقًا). وسوف نختار أيضًا ثابتًا a بحيث تكون القيمة العظمى للدالة المُوصوفة بالحد $O(n)$ المذكور آنفًا (والذي يَصف المركبة غير العَوْدية لزمن تنفيذ الخوارزمية) محدودًا بالقيمة an لجميع قيم $n > 0$. وباستخدام هذه الفرضية الاستقرائية، يكون لدينا:

$$\begin{aligned}
\mathrm{E}[T(n)] &\le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an \\
&= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\
&= \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an \\
&\le \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2} \right) + an \\
&= \frac{2c}{n} \left(\frac{n^2-n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an \\
&= \frac{c}{n} \left(\frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an
\end{aligned}$$

$$= c\left(\frac{3n}{4}+\frac{1}{2}-\frac{2}{n}\right)+an$$
$$\le \frac{3cn}{4}+\frac{c}{2}+an$$
$$= cn-\left(\frac{cn}{4}-\frac{c}{2}-an\right).$$

نحتاج، بغية استكمال البرهان، أن نبين أنه إذا كانت قيمة n كبيرة بقدرٍ كافٍ، فإن التعبير الأخير يساوي على الأكثر cn أو، على نحو مكافئ ، أنَّ $cn/4 - c/2 - an \ge 0$. فإذا أضفنا $c/2$ إلى كلا الطرفين وأخرجنا العامل المشترك n، نحصل على $n(c/4 - a) \ge c/2$. فإذا اخترنا الثابت c بحيث $c/4 - a > 0$، أي $c > 4a$، أمكننا تقسيم كلا الطرفين على $c/4 - a$، ونحصل على:

$$n \ge \frac{c/2}{c/4-a} = \frac{2c}{c-4a}.$$

وهكذا، إذا افترضنا أن $T(n) = O(1)$ لقيم n التي تحقق $n < 2c/(c-4a)$، فإن $\mathrm{E}[T(n)] = O(n)$. نستنتج أنه يمكننا إيجاد أيِّ إحصائيةِ ترتيب، وخصوصًا الوسط، بزمنٍ خطيٍّ متوقع إذا افترضنا أن العناصر متمايزة.

تمارين

1-2.9

بيّن أن RANDOMIZED-SELECT لا تُحدث أبدًا استدعاءً عَوْديًّا لصفيفة من 0 عنصرًا.

2-2.9

أثبت أن المتحول العشوائيَّ المؤشِّرَ X_k والقيمةَ $T(\max(k-1, n-k))$ مستقلان أحدهما عن الآخر.

3-2.9

اكتب نسخة تكرارية لـ RANDOMIZED-SELECT.

4-2.9

افترض أننا نستخدم RANDOMIZED-SELECT لاختيار العنصر الأصغر للصفيفة $A = \langle 3, 2, 9, 0, 7, 5, 4, 8, 6, 1 \rangle$. صِفْ متتاليةً من التجزئات التي تعطي أسوأ أداء لـ RANDOMIZED-SELECT.

3.9 الاختيار بزمن خطي في أسوأ الحالات

سندرس الآن خوارزمية اختيار زمن تنفيذها $O(n)$ في أسوأ الحالات. إن الخوارزمية SELECT، كما هو الحال في RANDOMIZED-SELECT، تَجِدُ العنصرَ المطلوب بتجزئة صفيفة الدخل عَوْديًّا. غير أننا نضمن هنا تفريقاً

جيداً للصفيفة أثناء تجزئتها. تَستخدم SELECT خوارزمية التجزئة الحتمية PARTITION المستَخدَمة في الفرز السريع (انظر المقطع 1.7)، ولكنها مُعدَّلة لتأخذ العنصر الذي تجري التجزئة حوله باعتباره موسط دخل.

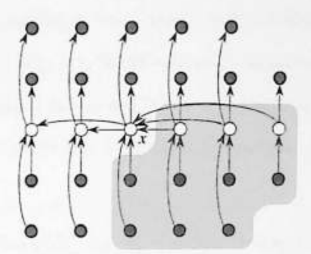
تحدد الخوارزمية SELECT العنصر الأصغر ذا الترتيب i لصفيفة دخل من $n > 1$ عنصرًا متمايزًا بتنفيذ الخطوات التالية. (إذا كان $n = 1$، فإن SELECT تعيد ببساطة قيمة الدخل الوحيد بوصفه العنصر الأصغر ذا الترتيب i.)

1. قسّم الـ n عنصرًا لصفيفة الدخل إلى $\lfloor n/5 \rfloor$ مجموعةً كلٌّ منها من 5 عناصر ومجموعةٍ واحدة على الأكثر تُنشئها من العناصر المتبقية من قسمة n على 5.
2. اكتشفِ الوسطَ لكلٍّ من المجموعات الـ $\lceil n/5 \rceil$ باستخدام فرز بالإدراج لفرز عناصر كل مجموعة (لكل منها 5 عناصر على الأكثر) ثم اخْتَرِ الوسطَ من اللائحة المفروزة لعناصر المجموعة.
3. استخدِمْ SELECT عَوْديًّا لاكتشاف الوسط x من الأوساط الـ $\lceil n/5 \rceil$ التي اكتشفتها في الخطوة 2. (إذا كان عدد الأوساط زوجيًّا، فإن x بحسب اصطلاحنا هو الوسط الأدنى.)
4. جَزِّئْ صفيفة الدخل حول وسط الأوساط x باستخدام نسخة مُعدلة من PARTITION. لتكن k أكبر بواحد من عدد العناصر في الجانب الأدنى من التجزئة، أي إن x هو العنصر الأصغر ذو الترتيب k ويوجد $n - k$ عنصرًا في الجانب الأعلى من التجزئة.
5. إذا كان $i = k$، فأعِدْ x. وإذا كان $i < k$، فاستخدِمْ SELECT عَوْديًّا لاكتشاف العنصر الأصغر ذي الترتيب i في الجانب الأدنى. وإذا كان $i > k$، فاستخدِمْ SELECT عَوْديًّا لاكتشاف العنصر الأصغر ذي الترتيب $i - k$ في الجانب الأعلى.

لتحليل زمن تنفيذ SELECT، نحدِّد بدايةً الحدَّ الأدنى لعدد العناصر التي هي أكبر من عنصر التجزئة x. يساعدنا الشكل 1.9 في إظهار هذه الخطوات. إن نصف الأوساط – على الأقل – التي اكتشفناها في الخطوة 2 أكبر من وسط الأوساط x أو تساويه[1]. ومنه، فإن نصف المجموعات الـ $\lceil n/5 \rceil$ على الأقل تتضمن 3 عناصر على الأقل أكبر من x، ما عدا المجموعتَيْن التاليتين: المجموعةِ التي عناصرها أقل من 5 عناصر في حال كانت n لا تقبل القسمة تمامًا على 5، والمجموعةِ التي تحتوي x نفسه. وباستبعاد هاتين المجموعتين، فإن عدد العناصر التي هي أكبر من x يساوي على الأقل

$$3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6 \; .$$

[1] بسبب افتراضنا أن العناصر متمايزة، فإن جميع الأوساط – ما عدا x – هي إما أكبر من x وإما أصغر من x.



الشكل 1.9 تحليل الخوارزمية SELECT. جرى تمثيل العناصر الـ n بدوائر صغيرة، وكل مجموعة من خمس عناصر تشغل عمودًا. لُوِّنت أواسط المجموعات بالأبيض، ووُضِعت لصاقة إلى جانب وسط الأوساط x. (عندما يكون الوسط لعددٍ زوجيٍّ من العناصر، فإننا نستخدم الوسط الأدنى.) تتجه الأسهم من العناصر الكبرى إلى العناصر الصغرى، والتي منها يمكن ملاحظة أن 3 من كل مجموعة ممتلئة بـ 5 عناصر إلى يمين x هي أكبر من x، وأن 3 من كل مجموعة من 5 عناصر إلى يسار x هي أقل من x. العناصر التي عُلِمَ أنها أكبر من x على تَظهر خلفية مظللة.

وبالمثل، فإن $3n/10 - 6$ عنصرًا على الأقل هي أقل من x. وهكذا، في أسوأ الحالات، فإن الخطوة 5 تستدعي SELECT عَوْديًا لـ $7n/10 + 6$ عنصرًا على الأكثر.

يمكننا الآن استنتاج علاقة عَوْدية لزمن التنفيذ في أسوأ الحالات $T(n)$ للخوارزمية SELECT. تستغرق الخطوات 1 و 2 و 4 زمنًا $O(n)$. (تتضمن الخطوة 2 $O(n)$ استدعاءً لفرز بالإدراج على مجموعات أحجامها $O(1)$.) تستغرق الخطوة 3 زمنًا $T(\lceil n/5 \rceil)$، وتستغرق الخطوة 5 على الأكثر زمنًا $T(7n/10 + 6)$، بافتراض أن T متزايدة باطراد. سنفترض فرضية، تبدو في البداية غير مبررة، وهي أنَّ أيَّ دخلٍ أقلُّ من 140 عنصرًا يحتاج إلى زمنٍ $O(1)$؛ سنوضح قريبًا منشأ هذا الثابت السحري 140. لذا يمكننا الحصول على العَوْدية

$$T(n) \le \begin{cases} O(1) & \text{if } n < 140 , \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n \ge 140 . \end{cases}$$

سنبين أن زمنَ التنفيذ خطيٌّ بالتعويض. وبعبارةٍ أدق، سنبين أن $T(n) \le cn$ لقيم الثابت c الكبيرة بقدر مناسب ولجميع قيم $n > 0$. سنبدأ بافتراض أن $T(n) \le cn$ لقيم الثابت c الكبيرة بقدر مناسب ولجميع قيم $n < 140$؛ وهذا الفرضية محقَّقةٌ إذا كان c كبيرًا بقدر كافٍ. سوف نحدد أيضًا الثابت a بحيث تكون قيمة الدالة الموصوفة بالحد $O(n)$ المذكور آنفًا (والذي يَصِفُ المكوِّنَ غير العَوْدي لزمن تنفيذ الخوارزمية) محدودًا بالقيمة an لجميع قيم $n > 0$. بتعويض هذا الفرضية الاستقرائية في الطرف الأيمن للعَوْدية نحصل على:

$$\begin{aligned} T(n) &\le c\lceil n/5 \rceil + c(7n/10 + 6) + an \\ &\le cn/5 + c + 7cn/10 + 6c + an \end{aligned}$$

$$= 9cn/10 + 7c + an$$
$$= cn + (-cn/10 + 7c + an) ,$$

الذي يساوي على الأكثر cn إذا كان

$$-cn/10 + 7c + an \leq 0 . \qquad (2.9)$$

إن المتراجحة (2.9) مكافئة للمتراجحة $c \geq 10a(n/(n-70))$ في حال $n > 70$. وحيث إننا افترضنا أن $n \geq 140$، يكون لدينا $n/(n-70) \leq 2$، وسيحقّق اختيارُنا $c \geq 20a$ المتراجحةَ (2.9). (لاحظ أنه لا يوجد شيءٌ خاصٌّ بالثابت 140؛ فبإمكاننا الاستعاضة عنه بأي عدد صحيح أكبر تمامًا من 70 ثم نختار c وفقًا له.) وعلى ذلك، فإنَّ زمنَ تنفيذ SELECT في أسوأ الحالات خطيٌّ.

إن الإجراءَيْن SELECT و RANDOMIZED-SELECT يحدِّدان، كما في الفرز بالمقارنة (انظر المقطع 1.8)، معلوماتٍ عن الترتيب النسبي للعناصر بمقارنة العناصر فقط. تذكَّر (من الفصل 8) أن الفرز يحتاج زمنًا $\Omega(n \lg n)$ في نموذج المقارنة، وحتى وسطيًا (انظر المسألة 8-1). تضع خوارزمياتُ الفرز بالزمن-الخطي في الفصل 8 افتراضاتٍ عن الدخل. وبالمقابل، لا تحتاج خوارزميات الاختيار بالزمن–الخطي في هذا الفصل إلى أي افتراضات عن الدخل. وهي لا تخضع للحد الأدنى $\Omega(n \lg n)$، لكونها قادرة على حل مسألة الاختيار من دون فرز. وهكذا، فإن حلَّ مسألةِ الاختيار باستخدام طريقة الفرز والفهرسة sorting and indexing، كما عُرِضَت في مقدمة هذا الفصل، هو حلٌّ غير فعَّالٍ على نحو مقارب.

تمارين

1-3.9

في خوارزمية SELECT، نقسم عناصر الدخل إلى مجموعاتٍ كلٌّ منها من 5 عناصر. هل تعمل الخوارزمية بزمن خطي لو قسمنا عناصر الدخل إلى مجموعاتٍ كلٌّ منها من 7 عناصر؟ ناقش أن SELECT لا تُنفَّذ بزمنٍ خطيٍّ إذا استخدمنا مجموعاتٍ كلٌّ منها من 3 عناصر.

2-3.9

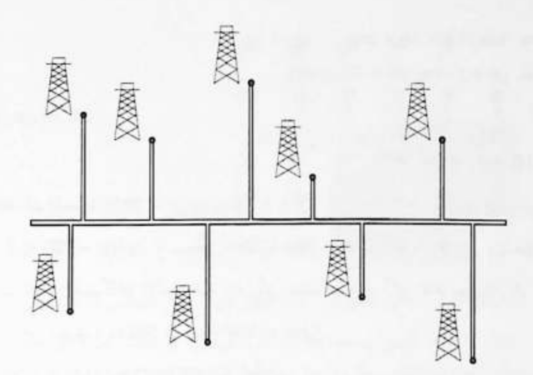
حلِّل الإجراء SELECT لتبين أنه في حال $n \geq 140$ ، فإن $\lceil n/4 \rceil$ عنصرًا على الأقل أكبرُ من وسط الأوساط x، وأنَّ $\lceil n/4 \rceil$ عنصرًا على الأقل أقلُّ من x.

3-3.9

بيّن كيف يمكن جعل الفرز السريع ينفَّذ بزمنٍ $O(n \lg n)$ في أسوأ الحالات، افترض أن جميع العناصر متمايزة.

★ *4-3.9*

افترض أن خوارزميةً تستخدم مقارناتٍ فقط لاكتشاف العنصر الأصغر ذي الترتيب i في مجموعة من n عنصرًا. بيّن أنه يمكن لهذه الخوارزمية أيضًا اكتشاف العنصر الأصغر ذي الترتيب $i-1$ والعنصر الأكبر ذي الترتيب $n-i$ دون إجراءِ أيِّ عملياتِ مقارنةٍ إضافية.



الشكل 2.9 يحتاج السيد أولي إلى تحديد موقع خط أنابيب نفط يتجه من الشرق إلى الغرب يجعل الطول الكلي للوصلات الفرعية المتجهة شمالاً أو جنوبًا أصغريًّا.

5-3.9

افترض أن لديك مساقًا فرعيًّا هو "صندوق-أسود black-box" لاكتشاف الوسط بزمن خطي في أسوأ الحالات. أعطِ خوارزميةً بسيطةً تَحلُّ مسألةَ الاختيار لإحصائية ترتيب اعتباطية بزمنٍ-خطي.

6-3.9

إن ***الكميات*** ذات الترتيب k (*k*th ***quantiles***) لمجموعةٍ من n عنصرًا هي إحصائيات الترتيب $k-1$ التي تَقسم مجموعةً مفروزةً إلى مجموعاتٍ متساوية-الحجم (باختلاف 1 على الأكثر). أعطِ خوارزميةً تسرد الكميات ذات الترتيب k لمجموعةٍ بزمن $O(n \lg k)$.

7-3.9

صِفْ خوارزميةً تُنفَّذ بزمنٍ $O(n)$ وتحدِّد (لمجموعةٍ معطاةٍ S من n عنصرًا متمايزًا وعددٍ صحيحٍ موجبٍ $k \le n$) الـ k عددًا من S التي هي أقرب ما يكون إلى وسط المجموعة S.

8-3.9

لتكن $X[1..n]$ و $Y[1..n]$ صفيفتان، تتضمن كلٌّ منهما n عنصرًا بترتيب مفروز سلفًا. أعطِ خوارزميةً، تنفَّذ بزمنٍ $O(\lg n)$، تكتشف وسط جميع عناصر $2n$ الصفيفتين X و Y.

9-3.9

السيد أولي Olay مستشار لشركة نفط تخطِّط لمدِّ خطٍّ أنابيب ضخم يتجه من الشرق إلى الغرب عبر حقل نفط فيه n بئرًا. ترغب الشركة في وصْل خطٍّ أنابيب فرعيٍّ مباشر من كل بئر إلى خط الأنابيب الرئيس عبر الطريق الأقصر (شمالاً أو جنوبًا)، كما هو مبين في الشكل 2.9. إذا أُعطينا الإحداثيات x و y للآبار، كيف

يمكن للسيد أولي اختيار الموقع الأمثل لخط الأنابيب الرئيس، الذي يجعل الطول الكلي للوصلات الفرعية أصغريًا؟ بيِّن كيف يمكن تحديد الموقع الأمثل بزمن خطي.

مسائل

1-9 أكبر i عددًا في ترتيب مفروز

لتكن لدينا مجموعة من n عددًا، ونرغب في اكتشاف أكبر i عددًا في ترتيبٍ مفروزٍ largest i numbers in sorted order باستخدام خوارزميةٍ تعتمد على المقارنات. أوجد خوارزميةً تنجز كلاً من الطرق الآتية بأفضل زمنِ تنفيذٍ مقارب في أسوأ الحالات، وحلِّل زمن تنفيذ الخوارزمية بدلالة n و i.

أ. افرز الأعداد واسرد أكبر i عددًا.

ب. ابنِ رتلاً ذا أولوية الأكبر max-priority queue من الأعداد، واستدعِ الإجراء EXTRACT-MAX i مرةً.

ت. استخدم خوارزمية إحصائية–ترتيب لاكتشاف العدد ذي الترتيب i في الكبر، جزِّئْ حول هذا العدد، وافرز أكبر i عددًا.

2-9 الوسط المثقَّل

إن ***الوسط (الأدنى) المثقَّل weighted (lower) median*** لـ n عنصرًا متمايزًا $x_1, x_2, \ldots, x_n$ ذات أوزان موجبة $\omega_1, \omega_2, \ldots, \omega_n$ تحقِّق $\sum_{i=1}^{n} \omega_i = 1$، هو العنصر x_k الذي يحقق:

$$\sum_{x_i < x_k} \omega_i < \frac{1}{2}$$

و

$$\sum_{x_i > x_k} \omega_i \le \frac{1}{2} .$$

فمثلاً، إذا كانت العناصر تساوي 0.1, 0.35, 0.05, 0.1, 0.15, 0.05, 0.2 وكل عنصر يساوي وزنه (أي، $\omega_i = x_i$ لكلِّ $i = 1, 2, \ldots, 7$)، فإن الوسط يساوي 0.1، في حين أن الوسط المُثقَّل يساوي 0.2.

أ. برهن أن وسط $x_1, x_2, \ldots, x_n$ هو الوسط المثقَّل لـ x_i بالأوزان $\omega_i = 1/n$ لكلِّ $i = 1, 2, \ldots, n$.

ب. بيِّن كيف نحسب الوسط المثقَّل لـ n عنصرًا بزمنٍ $O(n \lg n)$ في أسوأ الحالات باستخدام الفرز.

ت. بيِّن كيف نحسب الوسط المثقَّل بزمنٍ $\Theta(n)$ في أسوأ الحالات باستخدام خوارزمية الوسط بزمنٍ خطيٍّ linear-time median algorithm كالإجراء SELECT المذكور في المقطع 3.9.

نعرّف ***مسألة تحديد موقع مكتب البريد*** ***post-office location problem*** كما يلي. ليكن لدينا n نقطة $p_1, p_2, \ldots, p_n$ تَرتبط بها الأوزان $\omega_1, \omega_2, \ldots, \omega_n$. نرغب بإيجاد النقطة p (ليست بالضرورة أن تكون إحدى نقاط الدخل) التي تجعل المجموع $\sum_{i=1}^{n} \omega_i d(p, p_i)$ أصغريًا، حيث $d(a, b)$ المسافة بين النقطتين a و b.

ث. برهن أن الوسطَ المثقَّل هو الحلُّ الأفضل لمسألة تحديد موقعِ مكتبِ بريدٍ أحادي البعد 1-dimensional، بافتراض أن النقاطَ أعدادٌ حقيقية، والبعد بين النقطتين a و b يساوي $d(a, b) = |a - b|$.

ج. أوجد أفضل حلٍّ لمسألة تحديد موقعِ مكتبِ بريدٍ ثنائي البعد، بافتراض أن النقاطَ هي أزواجُ الإحداثيات (x, y)، والمسافة بين النقطتين $a = (x_1, y_1)$ و $b = (x_2, y_2)$ هي ***مسافة مانهاتن*** ***Manhattan distance*** التي تُعطى بالعلاقة $d(a, b) = |x_1 - x_2| + |y_1 - y_2|$.

9-3 *إحصائيات ترتيب صغير*

بيّنًا سابقًا أن عدد المقارنات في أسوأ الحالات $T(n)$ الذي يستخدمه الإجراء SELECT لاختيار الإحصائية ذات الترتيب i لـ n عددًا تحقق $T(n) = \Theta(n)$، ولكن الثابتَ المخفي ضمن تدوين-Θ كبيرٌ جدًا. فإذا كان i صغيرًا بالنسبة إلى n، يمكننا تنجيزُ إجراءٍ مختلفٍ يَستخدم SELECT باعتباره مساقًا فرعيًا ولكنه يُجري مقارناتٍ أقلَّ في أسوأ الحالات.

أ. صِفْ خوارزميةً تَستخدم $U_i(n)$ مقارنةً لاكتشاف العنصر الأصغر ذي الترتيب i لـ n عنصرًا، حيث:

$$U_i(n) = \begin{cases} T(n) & \text{if } i \geq n/2\ , \\ \lfloor n/2 \rfloor + U_i(\lceil n/2 \rceil) + T(2i) & \text{otherwise}\ . \end{cases}$$

(*تلميح*: ابدأ بـ $\lfloor n/2 \rfloor$ مقارنةً من الأزواج المنفصلة، وارجع عَوْديًا إلى المجموعة التي تحتوي على أصغر عنصر من كل زوج.)

ب. بيّن أنه إذا كان $i < n/2$ فإن $U_i(n) = n + O(T(2i)\lg(n/i))$.

ت. بيّن أنه إذا كان i ثابتًا أصغر من $n/2$ فإن $U_i(n) = n + O(\lg n)$.

ث. بيّن أنه إذا كان $i = n/k$ لكل $k \geq 2$، فإن $U_i(n) = n + O(T(2n/k)\lg k)$.

9-4 *تحليل بديل (آخر) لمسألة اختيارٍ ذات عشوائية مضافة*

سنستخدم في هذه المسألة متحولات عشوائية مؤشّرة لتحليل الإجراء RANDOMIZED-SELECT على نحو مماثل لتحليلنا لـ RANDOMIZED-QUICKSORT المذكور في المقطع 7.4.2.

سنفترض، كما في حالة تحليل الفرز السريع، أن جميع العناصر متمايزة، ونعيد تسمية عناصر صفيفة

الدخل A كما يلي: $z_1, z_2, \dots, z_n$، حيث z_i هو العنصر الأصغر ذو الترتيب i. وبذلك، يعيد الاستدعاء $\text{RANDOMIZED-SELECT}(A, 1, n, k)$ العنصرَ z_k.

ليكن:

$X_{ijk} = \mathrm{I}\{z_k$ is compared with z_j sometime during the execution of the algorithm to find $z_i\}$.

لكل $1 \le i < j \le n$،

أ. أعطِ تعبيرًا دقيقًا لـ $\mathrm{E}[X_{ijk}]$. (*تلميح*: يمكن أن يأخذ تعبيرك قيمًا مختلفة، تبعًا لقيم i و j و k.)

ب. لنرمز بـ X_k لعدد عمليات المقارنة الكلية بين عناصر الصفيفة A المُنفذة خلال عملية اكتشاف z_k. بيِّن أن:

$$\mathrm{E}[X_k] \le 2\left(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1}+\sum_{j=k+1}^{n}\frac{j-k-1}{j-k+1}+\sum_{i=1}^{k-2}\frac{k-i-1}{k-i+1}\right).$$

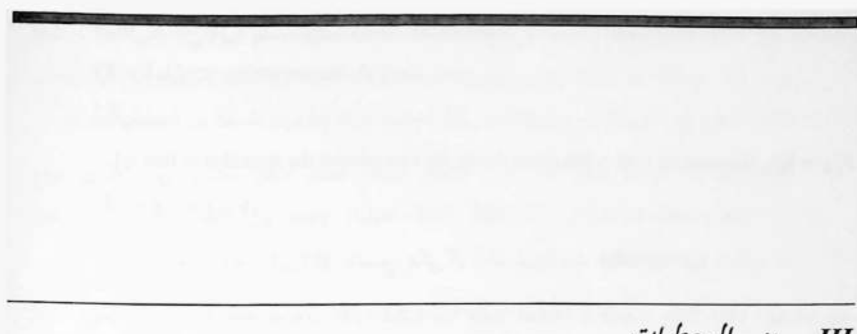
ت. بيِّن أن $\mathrm{E}[X_k] \le 4$.

ث. بافتراض أن جميع عناصر الصفيفة A متمايزة، استنتج أن RANDOMIZED-SELECT تُنفَّذ بزمن متوقع $O(n)$.

ملاحظات الفصل

ابتكَرَ Blum و Floyd و Pratt و Rivest و Tarjan [50] خوارزميةً تكتشف الوسط بزمنٍ خطيٍّ في أسوأ الحالات. وتعود أسرع نسخة ذات عشوائية مضافة إلى Hoare [169]. وكان Floyd و Rivest [108] قد طوَّرا نسخةً مُحسَّنة ذات عشوائية مضافة بحيث يجري اختيار التجزئات حول عنصر ما عَوْديًّا من عينةٍ صغيرةٍ من العناصر.

ما زال عدد المقارنات اللازمة لتحديد الوسط غير معروف بالضبط. غير أن Bent و John [41] قدَّما حدًّا أدنى يساوي $2n$ مقارنةً لاكتشاف الوسط، وقدَّم Schönhage و Paterson و Pippenger [302] حدًّا أعلى يساوي $3n$. وحسَّن Dor و Zwick هذين الحدَّيْن. وكان الحدُّ الأعلى الذي قدَّماه [94] أقلَّ بقليلٍ من $2.95n$، والحدَّ الأدنى [95] يساوي $(2+\epsilon)n$، حيث ϵ عدد ثابت موجب صغير، وبذلك حسَّنا قليلاً نتائج العمل الذي قام به Dor وآخرون [93]. وَصَفَ Paterson [272] بعضَ هذه النتائج إضافةً إلى أعمالٍ أخرى ذات صلةٍ بها.



III بنى المعطيات

تمهيد

تُعَدُّ المجموعاتُ sets من الأساسيات في علم الحاسوب مثلما هو شأنها في الرياضيات. وفي حين أن المجموعات في الرياضيات لا تتغير، فإن المجموعات التي تتعامل معها الخوارزميات يمكن أن تكبر أو تصغر أو تتغير مع الزمن، لذا فإننا نَصِفُ مثل هذه المجموعات بأنها ***ديناميكية dynamic***. تعرض الفصول الخمسة التالية بعض التقنيات الأساسية لتمثيل المجموعات الديناميكية المنتهية وكيفية التعامل معها حاسوبيًا.

قد تطلب الخوارزمياتُ تنفيذَ عدة أنواع من العمليات على المجموعات. على سبيل المثال، يحتاج العديد من الخوارزميات إلى إمكان إدراج عناصر في مجموعة، وحذف عناصر منها، واختبار الانتماء إليها. تسمَّى المجموعة الديناميكية التي تدعم هذه العمليات ***معجمًا dictionary***. وتطلب بعض الخوارزميات الأخرى عملياتٍ أكثرَ تعقيدًا. فمثلاً، تدعم أرتالُ ذات أولوية الأصغر أولاً - التي قدمناها في الفصل السادس ضمن سياق بنية المعطيات الكومة - عملياتِ إدراجِ عنصرٍ في مجموعة، واستخراج العنصر الأصغر في مجموعة. وتعتمد الطريقة الفضلى لتنجيز مجموعةٍ ديناميكيةٍ على العمليات التي يجب أن تقدّمها.

عناصر المجموعة الديناميكية

في التنجيز النموذجي للمجموعة الديناميكية، يمثِّل كلُّ عنصر بغرض، وفي حال وجود مؤشر يشير إلى هذا الغرض، يمكن فحص واصفاته والتعامل معها. (يناقش المقطع 3.10 تنجيز الأغراض والمؤشرات في بيئات البرمجة التي لا تحتوي هذه الأنماط باعتبارها أنماطَ معطياتٍ أساسية.) وتفترض بعض أنواع المجموعات الديناميكية أن أحد واصفات الغرض هو ***مفتاح key*** معرِّف للغرض. إذا كانت جميع المفاتيح مختلفة، يمكننا عندها أن نعتبر المجموعة الديناميكية مجموعةَ قيمٍ مفتاحية. وقد يحوي الغرض ***معطيات تابعة satellite data*** متضمنة في واصفات الغرض الأخرى، لكنها مع ذلك لا تُستخدم في تنجيز المجموعة. وقد تتضمن الأغراض كذلك واصفات يجري التعامل معها ضمن عمليات المجموعة، ويمكن أن تحتوي هذه الواصفات على معطيات

أو مؤشرات لأغراض أخرى في المجموعة.

تفترض بعض المجموعات الديناميكية سلفًا أن المفاتيح تنتمي إلى مجموعة مرتبة ترتيبًا كليًّا كمجموعة الأعداد الحقيقية أو مجموعة الكلمات المرتبة وفق الترتيب الأبجدي المألوف. يتيح لنا الترتيب الكلي تعريف العنصر الأصغر في المجموعة مثلاً، أو الحديث عن العنصر التالي الأكبر من عنصر محدد في المجموعة.

العمليات على المجموعات الديناميكية

يمكن تصنيف العمليات على المجموعات الديناميكية في صنفين: ***الاستفسارات queries*** التي تعيد ببساطة معلومات تتعلق بالمجموعة، ***وعمليات التعديل modifying operations*** التي تغيِّر المجموعة. وفيما يلي قائمة العمليات النموذجية على المجموعات. يحتاج أي تطبيق محدد عادة إلى تنجيز بضع عمليات منها فقط.

SEARCH(S, k)
استفسارٌ يأخذ مجموعةً معطاة S وقيمة المفتاح k، ويعيد مؤشرًا x إلى عنصر في S يحقق $x.key = k$، أو يعيد NIL إن لم يوجد في S مثل هذا العنصر.

INSERT(S, x)
عملية تعديل تضيف العنصر المشار إليه بـ x إلى المجموعة S. نفترض عادة أنه قد سبق إعطاء قيمة ابتدائية لجميع واصفات العنصر x التي يحتاجها تنجيز المجموعة.

DELETE(S, x)
عملية تعديل، تُزيل عنصر المجموعة المشار إليه بمؤشر x من المجموعة S (لاحظ أن هذه العملية تَستخدم مؤشرًا إلى عنصر x وليس قيمة مفتاح).

MINIMUM(S)
استفسارٌ يطبَّق على المجموعة S المرتبة ترتيبًا كليًّا، ويعيد مؤشرًا إلى العنصر الذي يمتلك المفتاح الأصغر في S.

MAXIMUM(S)
استفسارٌ يطبَّق على المجموعة S المرتبة ترتيبًا كليًّا، ويعيد مؤشرًا إلى العنصر الذي يمتلك المفتاح الأكبر في S.

SUCCESSOR(S, x)
استفسارٌ يأخذ عنصرًا x قيمةُ مفتاحِهِ موجودةٌ في المجموعة S المرتبة ترتيبًا كليًّا، ويعيد مؤشرًا إلى العنصر الأكبر منه مباشرة، أو يعيد NIL إذا كان x هو العنصر الأكبر في المجموعة.

PREDECESSOR(S, x)
استفسارٌ يأخذ عنصرًا x قيمةُ مفتاحِهِ موجودةٌ في المجموعة S المرتبة ترتيبًا كليًّا، ويعيد مؤشرًا إلى العنصر الأصغر منه مباشرة، أو يعيد NIL إذا كان x هو العنصر الأصغر في المجموعة.

في بعض الحالات، نستطيع توسيع الاستفسارَيْن SUCCESSOR و PREDECESSOR بحيث يمكن تطبيقهما على المجموعات التي تحوي عناصر غير متمايزة. ففي مجموعةٍ تتألف من n مفتاحًا، من الطبيعي أن نفترض سلفًا أن استدعاء MINIMUM متبوعًا بـ $n-1$ استدعاءً لـ SUCCESSOR يعدِّد عناصر المجموعة بالترتيب.

نقيس الزمن الذي يستغرقه تنفيذ عملية من عمليات المجموعة عادةً بدلالة حجم المجموعة. فمثلاً، يصف الفصل 13 بنية معطيات تدعم جميع العمليات المشار إليها آنفًا على مجموعةٍ حجمُها n بزمن $O(\lg n)$.

لمحة إلى الباب III

تصف الفصول من 10 إلى 14 عدة بنى معطيات يمكننا أن نستخدمها في تنجيز المجموعات الديناميكية. وسنستخدم العديد من هذه البنى لاحقًا لبناء خوارزميات فعالة في مسائل مختلفة، وقد عرضنا في الفصل السادس بنية معطيات هامة أخرى هي الكومة.

يقدم الفصل 10 أساسيات التعامل مع بنى المعطيات البسيطة كالمكدِّس، والرتل، والقائمة المترابطة، والشجرة ذات الجذر. ويبيِّن كذلك كيف تنجَّز الأغراض والمؤشرات في بيئات البرمجة التي لا تتضمن هذه البنى باعتبارها بنًى أولية. فإذا سبق أن دَرَستَ مقررًا في مقدمات البرمجة، فستجد أن محتويات هذا الفصل مألوفة لديك.

يعرِّف الفصل 11 جداول التلبيد hash tables التي تدعم العمليات المعجمية INSERT و DELETE و SEARCH. يحتاج التلبيد في أسوأ الحالات إلى زمن $\Theta(n)$ لإنجاز عملية SEARCH، غير أن الزمن المتوقع للعمليات الخاصة بجدول التلبيد هو $O(1)$. يستند تحليل التلبيد إلى الاحتمالات، غير أن غالبية الفصل لا تحتاج إلى خلفيةٍ عن هذا الموضوع.

تدعم أشجارُ البحث الثنائي التي يتناولها الفصل 12 جميعَ العمليات المتعلقة بالمجموعات الديناميكية الواردة آنفًا. وفي أسوأ الحالات تستغرق كلُّ عملية زمنًا $\Theta(n)$ في حالة شجرة ذات n عنصرًا، ولكن في حالة شجرة بحث ثنائي مبنية بناءً عشوائيًّا، يقدر زمن كل عملية بـ $O(\lg n)$. وتعتبر أشجار البحث الثنائي أساسًا للعديد من بنى المعطيات الأخرى.

يُعرِّف الفصل 13 الأشجار الحمراء–السوداء، وهي شكل مختلف من أشكال أشجار البحث الثنائي. وعلى العكس من أشجار البحث الثنائي العادية، تضمن الأشجار الحمراء–السوداء أداءها الجيد، فعملياتها تستغرق زمنًا $O(\lg n)$ في أسوأ الحالات. والشجرة الحمراء–السوداء هي شجرة بحث متوازنة. يقدم الفصل 18 في الباب V نوعًا آخر من الأشجار الثنائية المتوازنة يدعى B-tree (الأشجار المعممة). ومع أن آليات الأشجار الحمراء–السوداء معقَّدة نوعًا ما، إلا أنك تستطيع أن تقف على معظم خصائصها من الفصل دون دراسة آلياتها بالتفصيل. في جميع الأحوال، لا بد أنك ستجد أن التنقل ضمن الرماز مفيد جدًا.

في الفصل 14، نبيِّن كيف نُغْني الأشجار الحمراء–السوداء لدعم عمليات أخرى غير تلك العمليات الأساسية الواردة آنفًا. وسنُغْنيها أولاً بحيث نتمكن من دعم إحصائيات الترتيب ضمن مجموعة من المفاتيح، ثم نُغْنيها بطريقة أخرى لدعم مجالات الأعداد الحقيقية.

10 بنى المعطيات الأولية

نبحث في هذا الفصل تمثيلَ المجموعات الديناميكية ببنى معطياتٍ بسيطةٍ تَستخدم المؤشرات. ومع أننا نستطيع بناء العديد من بنى المعطيات المعقدة باستخدام المؤشرات، إلا أننا نقدم هنا البنى الأولية فقط وهي: المكدِّسات، والأرتال، واللوائح المترابطة، والأشجار ذات الجذر. نبيّن كذلك طرائق تركيب الأغراض والمؤشرات انطلاقًا من الصفيفات.

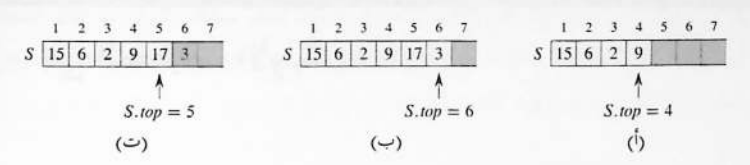
1.10 المكدِّسات والأرتال

المكدِّساتُ والأرتال مجموعاتٌ ديناميكية، تتميَّز بأن العنصر الذي يُحذف من المجموعة عند استخدام العملية DELETE عنصرٌ معروفٌ سلفًا. ففي ***المكدِّس stack*** يُحذف من المجموعة آخر عنصر أُدرج فيها، لذلك يقال بأن المكدِّس ينجِّز استراتيجيةَ ***الداخل آخرًا خارج أولاً last-in, first-out*** أو ***LIFO***. وبالمثل، يُحذف دومًا من ***الرتل queue*** العنصرُ الذي قضى أطول زمن فيه؛ أي إن الرتل ينجِّز استراتيجيةَ ***الداخل أولاً خارج أولاً first-in, first-out*** أو ***FIFO***. توجد عدة طرق فعّالة لتنجيز المكدِّسات والأرتال ضمن الحاسوب. نبيِّن في هذا المقطع كيف نستخدم صفيفةً بسيطةً لتنجيز كلٍّ من هاتين البنيتَيْن.

المكدِّسات

تسمى العملية INSERT في المكدِّسات غالبًا PUSH، وتسمى العملية DELETE التي لا تأخذ أيَّ موسطٍ POP. توحي هذه الأسماء بالمكدِّسات الحقيقية كمكدِّسات الصحون ذوات النوابض المستخدمة في المقاهي. إن ترتيب سحب الصحون من المكدِّس يعاكس ترتيب إدراجها في المكدِّس، نظرًا لأننا نسحب فقط الصحن الموجود في أعلى المكدِّس لأنه الوحيد المتاح للسحب.

يبين الشكل 1.10 تنجيز مكدِّس يتألف من n عنصرًا على الأكثر باستخدام صفيفة $S[1..n]$. للصفيفة واصفةٌ $S.top$ تعطي الدليل الموافق للعنصر المدرج آخرًا. يتألف المكدِّس من العناصر $S[1..S.top]$ حيث $S[1]$ هو العنصر الموجود في قعر المكدِّس، و $S[S.top]$ العنصر الموجود في قمة المكدِّس.



الشكل 1.10 تنجيز المكدّس S باستخدام صفيفة. تظهر عناصر المكدّس فقط في المواضع المظلّلة تظليلاً خفيفًا. **(أ)** المكدّس S وفيه 4 عناصر. العنصر الموجود في القمة هو 9. **(ب)** المكدّس S بعد الاستدعاءات PUSH(S, 17) و PUSH(S, 3). **(ت)** المكدّس S بعد أن أعاد الاستدعاء POP(S) العنصر 3 وهو العنصر الذي دُفع به أخيرًا في المكدّس. ومع أن العنصر 3 لا يزال يظهر في الصفيفة، إلا أنه لم يعد في المكدّس. فالعنصر الموجود في قمة المكدّس هو العنصر 17.

إذا كان $S.top = 0$، فإن هذا يعني أن المكدّس لا يحتوي على عناصر وهو ***فارغ empty***. يمكننا اختبار خلو المكدّس باستخدام عملية الاستفسار STACK-EMPTY. إذا حاولنا نزع عنصرٍ من مكدّس فارغ نقول عندها أن المكدّس ***يغيض underflows***، وهذا يعتبر بالطبع خطأً. وإذا تجاوز $S.top$ القيمة n فإن المكدّس ***يفيض overflows***. (في تنجيزنا الوارد ضمن شبه الرماز لا تهتم بحالة فيض المكدّس.)

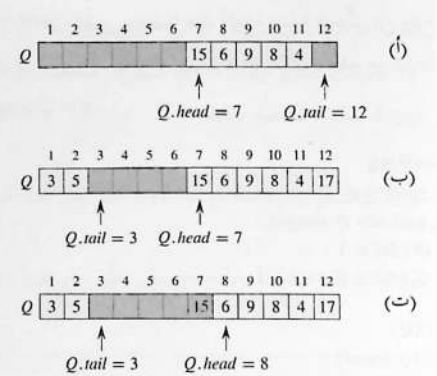
نستطيع تنجيز عمليات المكدّس ببضعة أسطر من الرماز.

```
STACK-EMPTY(S)
1  if S.top == 0
2      return TRUE
3  else return FALSE

PUSH(S, x)
1  S.top = S.top + 1
2  S[S.top] = x

POP(S)
1  if STACK-EMPTY(S)
2      error "underflow"
3  else S.top = S.top − 1
4      return S[S.top + 1]
```

يبين الشكل 1.10 آثار عمليتَي التعديل PUSH و POP. تستغرق كلُّ عمليةٍ من عمليات المكدّس الثلاث زمنًا $O(1)$.



الشكل 2.10 تنجيز الرتل باستخدام صفيفة $Q[1..12]$. تُظهر عناصرُ الرتل في المواضع المظلّلة تظليلاً خفيفًا فقط. **(أ)** في الرتل 5 عناصر، في المواضع $Q[7..11]$. **(ب)** تشكيلة الرتل بعد الاستدعاءات ENQUEUE$(Q, 17)$ و ENQUEUE$(Q, 3)$ و ENQUEUE$(Q, 5)$. **(ت)** تشكيلة الرتل بعد أن يعيد الاستدعاء DEQUEUE(Q) قيمة المفتاح 15 الذي كان سابقًا في رأس الرتل، وقد أصبحت القيمة الجديدة للرأس هي المفتاح 6.

الأرتال

نسمّي العمليةَ INSERT في الأرتال: ENQUEUE، ونسمّي العمليةَ DELETE في الأرتال: DEQUEUE. وكما في العملية POP في المكدّسات، لا تأخذ العملية DEQUEUE أي موسط. تجعل خاصيةُ FIFO الرتلَ يعمل كصفٍّ من الزبائن الذين ينتظرون التسديد أمام الصندوق. وللرتل ***رأس head* وذيل *tail***. عندما يُلحق عنصر بالرتل يأخذ مكانه في ذيل الرتل، تمامًا كما يحصل عندما يَصطفُّ الزبونُ الذي يصل في الآخر في نهاية الصف. أما العنصر الذي يُنزع من الرتل فهو دومًا العنصر الموجود في رأس الرتل، تمامًا كالزبون الموجود في أول الصف الذي كان أكثر الزبائن انتظارًا.

يبين الشكل 2.10 إحدى طرائق تنجيز رتلٍ من $n-1$ عنصرًا على الأكثر باستخدام صفيفة $Q[1..n]$. يزوَّد الرتل بواصفةٍ $Q.head$ تعطي الدليل الموافق لرأس الرتل أو تشير إليه. وتدل الواصفة $Q.tail$ على الموقع التالي الذي سيُدرَج فيه العنصر الجديد الواصل إلى الرتل. توجد عناصر الرتل في المواضع: $Q.head, Q.head+1, \dots, Q.tail-1$، حيث "نلتفّ" حول البداية بمعنى أن الموضع 1 يتبع مباشرة الموضع n في ترتيب دائري. فإذا أصبح $Q.head = Q.tail$، فهذا يعني أن الرتل فارغ. في البداية يكون $Q.head = Q.tail = 1$. فإذا كان الرتل فارغًا وحاولنا نزع عنصرٍ منه، فإن ذلك يتسبَّب في غيض الرتل. وإذا كان $Q.head = Q.tail + 1$ أو كان $Q.head = 1$ و $Q.tail = Q.length$ فهذا يعني أن الرتل ممتلئ، وأية محاولة لإلحاق عنصر به تتسبب في فيض الرتل.

في الإجراءَيْن: ENQUEUE و DEQUEUE اللذين نعرضهما هنا، أغفلنا عملية التحقُّق من وقوع الخطأ في حالتَي الغيض أو الفيض. (يُطلب إليك في التمرين 10.1-4 أن تضيف الرماز الذي يتحقق من الخطأ في هذين الشرطين.) يفترض شبه الرماز أن $n = Q.length$.

```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3      Q.tail = 1
4  else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
1  x = Q[Q.head]
2  if Q.head == Q.length
3      Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

يبين الشكل 10.2 نتائج العمليتَيْن ENQUEUE و DEQUEUE. وتستغرق كلٌّ منهما زمنًا $O(1)$.

تمارين

10.1-1

باستخدام الشكل 10.1 نموذجًا، أوضح نتيجة كلّ عملية وفق التتالي PUSH$(S, 4)$، PUSH$(S, 1)$، PUSH$(S, 3)$، POP(S)، PUSH$(S, 8)$، POP(S) على مكدِّسٍ S فارغٍ ابتداءً ومخزَّنٍ في الصفيفة $S[1..6]$.

10.1-2

اشرح كيفية تنجيز مكدِّسَيْن في صفيفة واحدة $A[1..n]$ بحيث لا يفيض أيٌّ منهما إلا في الحالة التي يبلغ فيها مجموع العناصر في المكدِّسين معًا n. ينبغي أن تنفَّذ العمليتان PUSH و POP خلال زمن $O(1)$.

10.1-3

باستخدام الشكل 10.2 نموذجًا، أوضح نتيجةَ كلّ عمليةٍ في المتتاليات ENQUEUE$(Q, 4)$ و ENQUEUE$(Q, 1)$ و ENQUEUE$(Q, 3)$ و DEQUEUE(Q) و ENQUEUE$(Q, 8)$ و DEQUEUE(Q) على رتلٍ Q فارغٍ ابتداءً ومخزَّن في الصفيفة $Q[1..6]$.

10.1-4

أعد كتابة ENQUEUE و DEQUEUE بحيث تكتشف حصول الغيض والفيض في الرتل.

10.1-5

رأينا أن المكدِّس يَسمح بإدراجِ العناصر وحذفِها من طرفٍ واحدٍ فقط، وأن الرتل يَسمح بالإدراج في أحد

الطرفَيْن والحذفَ من الطرف الآخر، أما ***deque*** (وهو رتلٌ ثنائي الطرفين) فإنه يَسمح بالإدراج والحذف من كلا الطرفين. اكتب أربعة إجراءاتٍ تستغرق زمنًا $O(1)$ لإدراج العناصر وحذفها من طرفيْ رتلٍ ثنائي الطرفين deque منجَّزٍ باستخدام صفيفة.

6-1.10
بين كيف ننجّز رتلاً باستخدام مكدِّسين. حلِّل زمن تنفيذ عمليات الرتل.

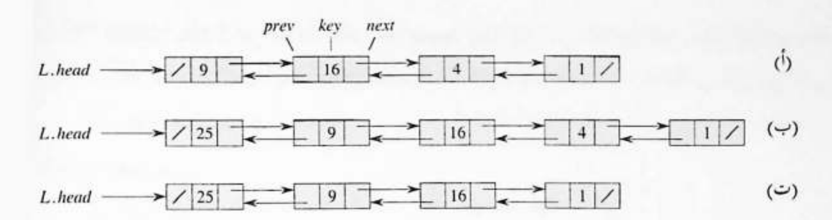
7-1.10
بيِّن كيف ننجّز مكدِّسًا باستخدام رتلَيْن. حلِّل زمن تنفيذ عمليات المكدِّس.

2.10 اللوائح المترابطة

اللائحةُ المترابطة بنيةُ معطياتٍ تُرتَّب فيها الأغراض ترتيبًا خطيًّا. ومع ذلك، تختلف اللائحة الخطية عن الصفيفة من جهةِ طريقةِ الترتيب؛ ففي الصفيفة يُحدِّد دليلُ الصفيفة الترتيبَ الخطيَّ، في حين يُحدَّد الترتيبُ في اللائحة المترابطة باستخدام مؤشرٍ في كل غرض. تقدِّم اللوائحُ المترابطة تمثيلاً بسيطًا ومرنًا للمجوعات الديناميكية، وتدعم جميع العمليات المشار إليها في الصفحة 230 (وإن لم تكن فعّالة بالضرورة).

وكما يَظهر في الشكل 3.10، فإن كلَّ عنصرٍ في ***لائحة مضاعفة الترابط doubly linked list L*** هو غرضٌ يمتلك الواصفةَ *key* وواصفتَيْن أخريَيْن هما: *التالي next والسابق prev*. ويمكن للغرض أن يحتوي أيضًا معطياتٍ تابعة أخرى. ليكن x عنصرًا في اللائحة، يشير $x.next$ إلى العنصر التالي في اللائحة المترابطة، ويشير $x.prev$ إلى العنصر السابق له فيها. فإذا كان $x.prev = \text{NIL}$ فهذا يعني أنه لا يوجد سابق للعنصر x، ومن ثَم فهو العنصر الأول في اللائحة أو ***رأس head*** اللائحة. وإذا كان $x.next = \text{NIL}$ فهذا يعني أنه لا يوجد عنصر يلي العنصر x، ومن ثَم فهو العنصر الأخير في اللائحة أو ***ذيل tail*** اللائحة. تشير الواصفة $L.head$ إلى العنصر الأول في اللائحة. فإذا كان $L.head = \text{NIL}$، فإن اللائحة تكون فارغة.

توجد عدة أشكال للوائح، فقد تكون اللائحة بسيطة الترابط أو مضاعفة الترابط، وقد تكون مرتبة أو غير مرتبة، وقد تكون دائرية أو غير دائرية. فإذا كانت اللائحة ***بسيطة الترابط singly linked*** فإننا نلغي المؤشر *prev* في كل عنصر. وإذا كانت اللائحة ***مرتبة sorted*** يكون الترتيب الخطي للائحة مطابقًا للترتيب الخطي للمفاتيح المخزنة في عناصر اللائحة؛ وعندها يكون العنصر الأصغر رأسًا للائحة، والعنصرُ الأكبر ذيلاً للائحة. وإذا كانت اللائحة ***غير مرتبة unsorted*** فيمكن أن تَظهر العناصر في أي ترتيب. أما إذا كانت ***اللائحة دائرية circular list*** فإن المؤشر *prev* في رأس اللائحة يشير إلى الذيل، والمؤشر *next* في ذيل اللائحة يشير إلى الرأس. وهكذا يمكن النظر إلى اللائحة وكأنها حلقة من العناصر. سنفترض في بقية هذا المقطع أن اللوائح التي نتعامل معها هي لوائح غير مرتبة ومضاعفة الترابط.



الشكل 3.10 (أ) لائحة مضاعفة الترابط L تمثل المجموعة الديناميكية $\{1,4,9,16\}$. كل عنصر من اللائحة هو غرضٌ يمتلك واصفةً للمفتاح وواصفتين للمؤشرين (يظهران على شكل أسهم) على الغرضين السابق والتالي. إن قيمة الواصفة *next* في الذيل والواصفة *prev* في الرأس هي NIL ويشار إليها بخط مائل. تشير الواصفة *L.head* إلى الرأس. **(ب)** بعد تنفيذ العملية LIST-INSERT(L,x) حيث $x.key = 25$ يصبح في اللائحة المترابطة غرضٌ جديد قيمة مفتاحه 25 ويأخذ مكانه كرأس جديد للائحة. يشير الغرض الجديد إلى الرأس القديم ذي القيمة 9. **(ت)** النتيجة المترتبة عن الاستدعاء LIST-DELETE(L,x) حيث يشير x إلى الغرض ذي القيمة 4.

البحث في اللائحة المترابطة

يعمل الإجراء LIST-SEARCH(L,k) على إيجاد أول عنصر يحوي المفتاح k في اللائحة L وذلك بإجراء بحث خطي بسيط، ويعيد مؤشرًا إلى هذا العنصر. إن لم يجد الإجراءُ أيَّ غرضٍ في اللائحة يحوي المفتاح k، فإنه يعيد NIL. بالعودة إلى اللائحة المترابطة في الشكل 3.10(أ)، يعيد الاستدعاءُ LIST-SEARCH$(L,4)$ مؤشرًا إلى العنصر الثالث، ويعيد الاستدعاءُ LIST-SEARCH$(L,7)$ النتيجة NIL.

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3      x = x.next
4  return x
```

يستغرق البحث في لائحةٍ تتألف من n غرضًا باستخدام الإجراء LIST-SEARCH زمنًا $\Theta(n)$ في أسوأ الحالات، نظرًا لأنه قد يحتاج إلى البحث في كامل اللائحة.

الإدراج في اللائحة المترابطة

ليكن لدينا العنصر x الذي وُضعت في واصفته *key* قيمةٌ معينة. يعمل الإجراء LIST-INSERT على "لصق" العنصر x في مقدمة اللائحة المترابطة كما يَظهر في الشكل 3.10(ب).

```
LIST-INSERT(L, x)
1  x.next = L.head
2  if L.head ≠ NIL
```

```
3        L.head.prev = x
4   L.head = x
5   x.prev = NIL
```

(تذكَّر أن التدوينَ المستخدَم للواصفات يمكن أن يكون متسلسلاً، بحيث تعبِّر $L.head.prev$ عن الواصفة $prev$ في الغرض الذي يشير إليه $L.head$.) إن زمن تنفيذ LIST-INSERT في لائحةٍ تتألف من n عنصرًا هو $O(1)$.

الحذف من اللائحة المترابطة

يُزيل الإجراءُ LIST-DELETE عنصرًا x من اللائحة المترابطة L. ويجب أن يأخذ هذا الإجراء مؤشرًا إلى x ثم "يفك لصقه" من اللائحة ويحدِّث المؤشرات. إذا كنا نرغب في حذفِ عنصرٍ ذي مفتاحٍ محدَّد، فيجب أن نستدعي أولاً LIST-SEARCH لاسترداد المؤشر إلى هذا العنصر.

```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2        x.prev.next = x.next
3   else L.head = x.next
4   if x.next ≠ NIL
5        x.next.prev = x.prev
```

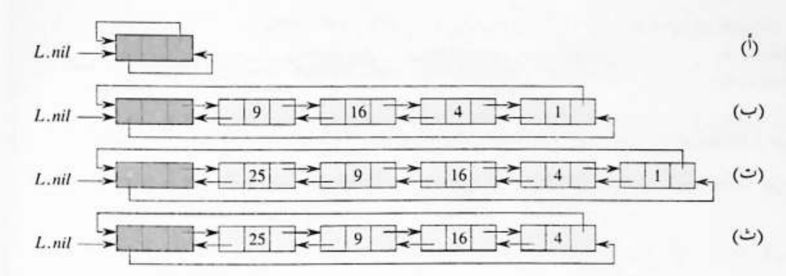
يبيِّن الشكل 3.10(ت) كيف يُحذف عنصر من اللائحة المترابطة. يعمل LIST-DELETE في زمن $O(1)$. ولكن إذا كنا نرغب بحذف عنصرٍ ذي مفتاحٍ محدَّد، فيلزمنا $\Theta(n)$ في أسوأ الحالات لأننا يجب أن نستدعي LIST-SEARCH أولاً لاكتشاف العنصر.

الحَرَس

يمكن أن يصبح رماز الإجراء LIST-DELETE أبسط إذا استطعنا تجاهل الشروط الحدية على رأس وذيل اللائحة.

```
LIST-DELETE′(L, x)
1   x.prev.next = x.next
2   x.next.prev = x.prev
```

الحارس ***sentinel*** غرضٌ شكليٌّ يمكِّننا من تبسيط الشروط على الحدود. لنفترض مثلاً أننا نزوِّد اللائحةَ L بغرضٍ $L.nil$ يمثِّل NIL، ولكنه يمتلك جميع الواصفات التي تمتلكها بقية عناصر اللائحة. وأينما ورد ذكر NIL في رماز اللائحة، نستعيض عنه بالحارس $L.nil$. يبين الشكل 4.10 كيف تتحول لائحة مضاعفة الترابط عادية إلى ***لائحة دائرية، مضاعفة الترابط مزودة بحارس*** ***circular, doubly linked list with a sentinel***،



الشكل 4.10 لائحة دائرية مضاعفة الترابط مزودة بحارس. يَظهر الحارس *L.nil* بين الرأس والذيل. لم تعد هناك حاجة للواصفة *L.head*، لأننا يمكن أن ننْفُذَ إلى رأس اللائحة عبر *L.nil.next*. **(أ)** لائحة فارغة. **(ب)** اللائحة المترابطة الواردة في الشكل 10.3(أ)، مع المفتاح 9 في الرأس والمفتاح 1 في الذيل. **(ت)** اللائحة بعد تنفيذ LIST-INSERT′(L,x)، حيث $x.key = 25$. وقد أصبح الغرض الجديد رأسًا للائحة. **(ث)** اللائحة بعد حذف الغرض ذي المفتاح 1، وقد أصبح الغرضُ ذو المفتاح 24 ذيلَ اللائحة الجديد.

حيث يوضع الحارس *L.nil* بين الرأس والذيل. تشير الواصفة *L.nil.next* إلى رأس اللائحة، وتشير *L.nil.prev* إلى ذيلها. وبالمثل، فإن كلاً من الواصفتين *next* الخاصة بالذيل و *prev* الخاصة بالرأس تشير إلى *L.nil*. ولمّا كان *L.nil.next* يشير إلى الرأس، يمكننا حذف الواصفة *L.head* حذفًا كاملاً، والاستعاضة عنها أينما ورد ذكرها بـ *L.nil.next*. يبيّن الشكل 4.10(أ) أن اللائحة الفارغة تتألف من الحارس فقط، وأن كلاً من *L.nil.next* و *L.nil.prev* يشير إلى *L.nil*.

يبقى رماز LIST-SEARCH كما كان سابقًا، مع تبديل NIL و *L.head* أينما وردا كما ذكرنا آنفًا.

```
LIST-SEARCH′(L, k)
1  x = L.nil.next
2  while x ≠ L.nil and x.key ≠ k
3      x = x.next
4  return x
```

نستخدم الإجراء LIST-DELETE′ المؤلَّف من سطرين لحذف عنصرٍ من اللائحة. ونستخدم الإجراء التالي لإدراج عنصرٍ في اللائحة.

```
LIST-INSERT′(L, x)
1  x.next = L.nil.next
2  L.nil.next.prev = x
3  L.nil.next = x
4  x.prev = L.nil
```

يبين الشكل 4.10 أثر LIST-INSERT′ و LIST-DELETE′ على عيّنةٍ من اللوائح.

نادرًا ما يخفِّض الحرسُ الحدودَ المقارِبة لزمن العمليات على بنى المعطيات، ولكنه قد يُخفِّض المعاملات الثابتة. وينحصر الكسب الناتج عن استخدام الحرس في الحلقات عادةً في وضوح الرماز لا في السرعة. فمثلاً، يُبسِّطُ استخدامُ الحرس الرمازَ الخاصَّ باللائحة المترابطة، ولكننا نوفِّر هنا زمنًا $O(1)$ فقط في الإجراءَيْن LIST-INSERT′ و LIST-DELETE′. غير أن استخدام الحرس في حالات أخرى يساعد على إحكام الرماز في الحلقة، وهذا يُخفِّض أمثال n أو n^2 في زمن التنفيذ.

يجب أن نَستخدم الحرس بحذر، فإذا كانت لدينا عدة لوائح صغيرة، قد يشكِّل الخَزْن الإضافي الذي يُستخدم لحرس هذه اللوائح ضَياعًا مهمًا في الذاكرة. في هذا الكتاب، نَستخدم الحرس فقط عندما نجد أنه يبسِّط الرماز تبسيطًا حقيقيًّا.

تمارين

1-2.10

هل يمكن تنجيز عملية INSERT الخاصة بالمجموعات الديناميكية في اللوائح المترابطة البسيطة بزمن $O(1)$؟ وماذا عن DELETE؟

2-2.10

نَجِّزْ مكدِّسًا باستخدام لائحة مترابطة بسيطة L. يجب أن يبقى زمن العمليتين PUSH و POP $O(1)$.

3-2.10

نَجِّزْ رتلاً باستخدام لائحة مترابطة بسيطة L. يجب أن يبقى زمن العمليتين ENQUEUE و DEQUEUE $O(1)$.

4-2.10

مَرَّ معنا آنفًا أن كلَّ تكرارٍ للحلقة في الإجراء LIST-SEARCH′ يحتاج إلى اختبارين: أحدهما للتأكد أن $x \neq L.nil$، والآخر للتأكد أن $x.key \neq k$. بيِّن كيف يمكننا حذف اختبار $x \neq L.nil$ في كل تكرار.

5-2.10

نَجِّزْ العمليات المعجمية INSERT و DELETE و SEARCH مستخدمًا لوائح مترابطة بسيطة ودائرية. ما هي أزمنة تنفيذ هذه الإجراءات؟

6-2.10

تأخذ عملية UNION الخاصة بالمجموعات الديناميكية مجموعتَيْن S_1 و S_2 منفصلتين دخلاً لها، وتعيد مجموعةً $S = S_1 \cup S_2$ مؤلَّفةً من جميع عناصر S_1 و S_2. تُهدِّم هذه العملية المجموعتين S_1 و S_2 عادةً. بيِّن كيف تتحمّل UNION زمنًا $O(1)$ باستخدام لائحةٍ مناسبةٍ باعتبارها بنية معطيات.

7-2.10

اكتب إجراءً غير عَوْدي يَقلب لائحةً مترابطةً بسيطةً مؤلَّفةً من n عنصرًا بزمن $\Theta(n)$. يجب ألاَّ يَستخدم هذا الإجراء أماكن ثابتة للخزن أكثر مما تحتاجه اللائحة نفسها.

★ *8-2.10*

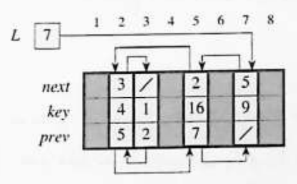
اشرح كيف تنجِّز اللوائحَ المترابطةَ المضاعفةَ مستخدمًا فقط قيمةً واحدةً من نوع مؤشر $x.np$ لكل عنصر بدلاً من استخدام المؤشرين المعتادين ($next$ و $prev$). افترض أن جميع قيم المؤشرات تكافئ أعدادًا صحيحة ممثَّلة في k خانة ($k-$bit)، وعرِّف $x.np$ بحيث يكون $x.np = x.next$ XOR $x.prev$، أي "الخيار المقصور" لـ k خانة ثنائية بين $x.next$ و $x.prev$. (يعبّر 0 عن القيمة NIL.) تأكَّد أنك وَصَفتَ المعلوماتِ اللازمةَ للنفاذ إلى رأس اللائحة. بيِّن كيف تنجِّز العمليات SEARCH و INSERT و DELETE على هذه اللائحة، وكيف تُقلَب هذه اللائحة بزمن $O(1)$.

3.10 تنجيز المؤشرات والأغراض

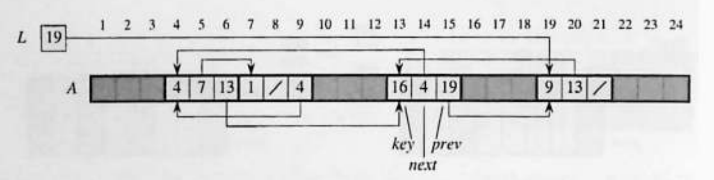
كيف ننجّز المؤشرات والأغراض في اللغات التي ليس لديها مثل هذه الأنماط؟ سنرى في هذا المقطع طريقتَيْن لتنجيز بنى المعطيات المترابطة دون الاستخدام الصريح لنمط معطيات مؤشر. سنركِّب الأغراض والمؤشرات انطلاقًا من الصفيفات وأدلة الصفيفات.

تمثيل الأغراض بعدة صفيفات

نستطيع تمثيل مجموعة أغراض تمتلك الواصفات ذاتها باستخدام صفيفة لكل واصفة. على سبيل المثال، يبين الشكل 5.10 تنجيز اللائحة المترابطة الواردة في الشكل 3.10(أ) باستخدام ثلاث صفيفات. تحتفظ الصفيفة key بقيم المفاتيح الموجودة حاليًا في المجموعة الديناميكية، وتخزَّن المؤشرات في الصفيفتين $next$ و $prev$. تمثِّل عناصرُ الصفيفات $key[x]$ و $next[x]$ و $prev[x]$ غرضًا واحداً في اللائحة المترابطة، وذلك لكل دليلِ صفيفةٍ معطًى x. وبموجب هذا التفسير، يكون المؤشر x ببساطة دليلاً مشتركًا للصفيفات key، و $next$ و $prev$.



الشكل 5.10 اللائحة المترابطة الواردة في الشكل 3.10(أ) ممثلة بالصفيفات key و $next$ و $prev$. تمثِّل كل شريحة عمودية من الصفيفات غرضًا واحدًا. تتوافق المؤشرات المخزَّنة مع أدلة الصفيفات المبينة في الأعلى. وتوضح الأسهم كيف تفسَّر هذه الأدلة. تحتوي المواضع المظللة تظليلاً خفيفًا عناصر اللائحة. ويحتفظ المتحول L بالدليل الذي يدل على الرأس.



الشكل 6.10 اللائحة المترابطة الواردة في الشكلين 3.10(أ) و 5.10 ممثَّلة في صفيفة وحيدة A. كل عنصر في اللائحة هو غرضٌ يَشغل صفيفةً جزئية متلاصقة طولها 3 ضمن الصفيفة. توافق الواصفات الثلاثة key و $next$ و $prev$ الزيحانات 0 و 1 و 2 على الترتيب ضمن كلّ غرض. إن مؤشرًا إلى الغرض هو دليلُ العنصر الأول في الغرض. ظُلِّلت الأغراضُ المحتوية على عناصر اللائحة تظليلاً خفيفًا، وتبين الأسهم الترتيب في اللائحة.

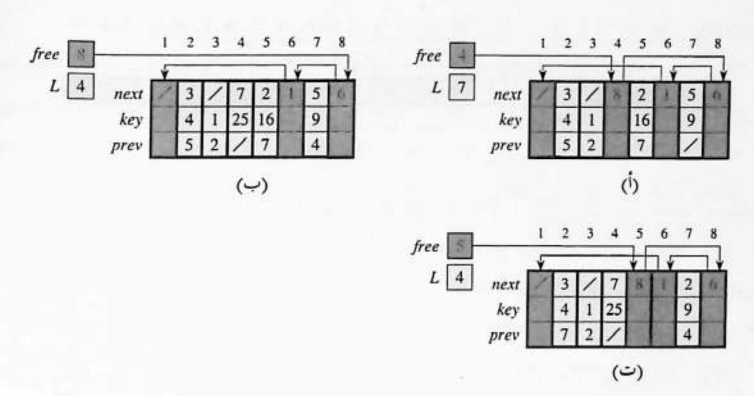
في الشكل 3.10(أ) يَتبع الغرضُ الذي يَحمل القيمة 4 الغرضَ الذي يَحمل القيمةَ 16 في اللائحة المترابطة. أما في الشكل 5.10، فتَظهر القيمةُ 4 في $key[2]$ و القيمة 16 في $key[5]$، وكذلك $next[5] = 2$ و $prev[2] = 5$. ومع أن الثابت NIL يَظهر في الواصفة $next$ في ذيل اللائحة وفي الواصفة $prev$ في رأسها، فإننا نستخدم عادةً عددًا صحيحًا (مثل 0 أو 1-) لا يمكن أن يعبِّر عن دليلٍ فعليٍّ في الصفيفات. يُستخدم متحولٌ L لحفظ الدليل الخاص برأس اللائحة.

تمثيل الأغراض بصفيفة وحيدة

تُعَنْوَن الكلمات في ذاكرة الحواسيب عادةً باستخدام أعدادٍ صحيحةٍ تقع قيمها بين 0 و $M - 1$، حيث M عدد صحيح كبير مناسب. وفي العديد من لغات البرمجة يَشغل الغرضُ مجموعةً من المواقع المتلاصقة في ذاكرة الحاسوب. ويحتوي المؤشر على عنوان موضع الذاكرة الأول من الغرض، أما المواضع الأخرى في الغرض نفسه فيمكن الحصول على دليلها بإضافة زيحان إلى المؤشر.

يمكننا أن نستخدم الاستراتيجية ذاتها لتنحيز الأغراض في بيئات البرمجة التي لا توفِّر صراحةً أنماط معطيات مؤشر. فمثلاً يوضح الشكل 6.10 كيف يمكننا أن نستخدم صفيفةً وحيدةً A لتخزين اللائحة المترابطة التي رأيناها في الأشكال 3.10(أ) و 5.10. يَشغل الغرضُ صفيفةً جزئيةً متلاصقة $A[j\,..\,k]$. ويوافق كلَّ واصفةٍ من الغرض زيحانٌ تقع قيمته بين 0 و $k - j$، ومؤشرٌ على الغرض هو الدليل j. في الشكل 6.10 تكون قيم الزيحان الموافقة لـ key و $next$ و $prev$ هي 0 و 1 و 2 على الترتيب. ولقراءة قيمة $i.prev$ اعتمادًا على مؤشر i معطى، نضيف القيمة i الخاصة بالمؤشر إلى قيمة الزيحان 2، أي إننا نقرأ $A[i + 2]$.

يعتبر التمثيل بصفيفة واحدة مرنًا من جهة أنه يسمح بتخزين الأغراض ذات الأطوال المختلفة في الصفيفة نفسها. إن مسألة إدارة مجموعة الأغراض غير المتجانسة هذه أصعب من مسألة إدارة مجموعة



الشكل 7.10 أثر الإجراءين ALLOCATE-OBJECT و FREE-OBJECT. (أ) اللائحة الواردة في الشكل 5.10 (مظلَّلة تظليلاً خفيفًا) واللائحة الحرة (مظللة تظليلاً ثقيلاً). تبيِّن الأسهمُ بنيةَ اللائحة الحرة. (**ب**) نتيجة استدعاء ALLOCATE-OBJECT() (التي تعيد الدليل 4)، حيث تضع 25 في *key*[4] وتستدعي LIST-INSERT(*L*, 4). إن الرأس الجديد للائحة الحرة هو الغرض 8 الذي كان موافقًا لـ *next*[4] في اللائحة الحرة. (**ت**) بعد تنفيذ LIST-DELETE(*L*, 5) نستدعي FREE-OBJECT(5). يصبح الغرضُ 5 الرأسَ الجديد للائحة الحرة، ويكون الغرضُ 8 التاليَ له في اللائحة الحرة.

متجانسة من الأغراض تمتلك فيها جميع الأغراض الواصفات نفسها. ولما كانت معظم بنى المعطيات التي سندرسها تتألف من عناصر متجانسة، فيكفينا أن نستخدم تمثيل الأغراض بعدة صفيفات لتحقيق هدفنا.

تحصيص الأغراض وتحريرها

لكي نتمكن من إدارج مفتاح في مجموعة ديناميكية ممثَّلة بلائحة مضاعفة الترابط، يجب أن نحصِّص مؤشرًا لغرضٍ غير مستخدَم حاليًا في تمثيل اللائحة المترابطة. لذا، يكون من المفيد إدارةُ خزْنِ الأغراض غير المستخدمة حاليًا في تمثيل اللائحة المترابطة بحيث نستطيع تحصيص أحدها عند الحاجة. تَستخدم بعضُ الأنظمة ***جامع نفايات*** ***garbage collector*** ليكون مسؤولاً عن تحديد الأغراض غير المستخدمة، في حين تتحمَّل العديدُ من الأنظمة البسيطة مسؤوليةَ إعادة الأغراض غير المستخدَمة إلى المدير المسؤول عن الخزن. نستكشف هنا مسألة تحصيص وتحرير (أو فكّ تحصيص) الأغراض المتجانسة باستخدام مثال عن لائحة مضاعفة الترابط ممثلة باستخدام عدة صفيفات.

لنفترض أن طول الصفيفات في التمثيل باستخدام عدة صفيفات هو m، وأن المجموعة الديناميكية تحتوي، في لحظة ما، $n \leq m$ عنصرًا. إن هذا يعني أن هناك n غرضًا تمثل العناصر الموجودة حاليًا في المجموعة

free 10

L_2 9

L_1 3

	1	2	3	4	5	6	7	8	9	10
next	5	/	6	8	/	2	1	/	7	4
key	k_1	k_2	k_3		k_5	k_6	k_7		k_9	
prev	7	6	/		1	3	9		/	

الشكل 8.10 لائحتان مترابطتان L_1 (مظلَّلة تظليلاً خفيفًا) و L_2 (مظللة تظليلاً ثقيلاً) ولائحة حرة مرافقة لهما (الأشدّ تظليلاً).

الديناميكية، أما الأغراض المتبقية وعددها $m - n$ غرضًا فهي ***حرة free***. يمكن أن تُستخدم الأغراض الحرة لتمثيل العناصر التي ستُدرَج في المجموعة الديناميكية لاحقًا.

نحتفظ بالأغراض الحرة في قائمة أحادية الترابط نسميها ***اللائحة الحرة free list***. تَستخدم اللائحة الحرة الصفيفة $next$ فقط، التي تخزِّن المؤشرات $next$ ضمن اللائحة. يحوي المتحول العام $free$ القيمةَ الدالةَ على رأس اللائحة الحرة. وعندما تكون المجموعةُ الديناميكية التي تمثلها اللائحة المترابطة L غيرَ فارغة، يمكن أن تكون اللائحة الحرة ملازمةً للائحة L كما يَظهر في الشكل 7.10. لاحظ أن كلَّ غرضٍ في التمثيل إما أن يكون في اللائحة L وإما في اللائحة الحرة، ولا يمكن أن يكون فيهما معًا.

تتصرَّف اللائحةُ الحرة كمكدِّس: فالغرض التالي المحصّص هو آخرُ غرضٍ سبق تحريره. ويمكننا أن ننجِّز عمليتَي المكدِّس PUSH و POP باستخدام اللائحة وذلك لتنجيز الإجراءَيْن الخاصين بتحصيص الأغراض وتحريرها على الترتيب. نفترض أن المتحول العام $free$ المستخدَم في الإجراءين التاليين يشير إلى العنصر الأول من اللائحة الحرة.

```
ALLOCATE-OBJECT()
1  if free == NIL
2       error "out of space"
3  else x = free
4       free = x.next
5       return x
```

```
FREE-OBJECT(x)
1  x.next = free
2  free = x
```

في البداية، تحوي اللائحةُ الحرة جميعَ الأغراض غير المحصّصة وعددها n. وعندما تُستَنْفَذ اللائحة الحرة، يُعلِن الإجراء ALLOCATE-OBJECT حدوثَ خطأ. يمكننا استخدام لائحةٍ حرةٍ وحيدة لتخدم عدة لوائح مترابطة. يبيِّن الشكل 8.10 لائحتَيْن مترابطتين ولائحةً حرةً مرافقةً لهما عبر الصفيفات key و $next$ و $prev$.

يُنفَّذ الإجراءان في زمن $O(1)$، وهو ما يجعلهما عَمَليَّيْن تمامًا. ويمكن تعديلهما ليعملا مع أية مجموعةٍ متجانسةٍ من الأغراض، وذلك بجعل أيٍّ من واصفات الأغراض يتصرَّف مثل الواصفة $next$ في اللائحة الحرة.

تمارين

1-3.10

ارسم شكلاً لمتتالية العناصر: $\langle 13, 4, 8, 19, 5, 11 \rangle$ المخزَّنة على أنها لائحةٌ مضاعفةُ الترابط باستخدام تمثيلٍ بعدة صفيفات، ثم كرِّر الرسمَ باستخدام تمثيلٍ بصفيفةٍ وحيدة.

2-3.10

اكتب الإجراءَيْن ALLOCATE-OBJECT و FREE-OBJECT في حالة مجموعة متجانسة من الأغراض، منجَّزة باستخدام تمثيلٍ بصفيفةٍ وحيدة.

3-3.10

لماذا لا نحتاج إلى وضع قيمة (أو تهيئة) الواصفات $prev$ في الأغراض عند تنجيز الإجراءَيْن ALLOCATE-OBJECT و FREE-OBJECT؟

4-3.10

في معظم الحالات، نرغب بأن تكون جميعُ عناصر اللائحة مضاعفةَ الترابط متراصَّةً compact في مكان الخزن، فنستخدم مثلاً المواقع ذات الأدلة m الأولى من التمثيل باستخدام عدة صفيفات. (وهذه هي حالة بيئة الحوسبة ذات الذاكرة الافتراضية الصَّفْحيّة paged virtual-memory computing enviroment.) اشرح كيف يُنجَّز الإجراءان ALLOCATE-OBJECT و FREE-OBJECT بحيث يكون التمثيل متراصًّا. افترض أنه لا توجد مؤشرات إلى عناصر من اللائحة المترابطة خارج اللائحة نفسها (*تلميح*: استخدم تنجيز المكدِّس باستخدام الصفيفة.)

5-3.10

لتكن L لائحةً مضاعفة طولها n مخزنةً في الصفيفات key و $prev$ و $next$ التي يبلغ طولها m. لنفترض أن الإجراءَيْن ALLOCATE-OBJECT و FREE-OBJECT يديران هذه الصفيفات، وأنهما يحتفظان بلائحةٍ حرةٍ مضاعفةِ الترابط F. ولنفترض كذلك أنه من بين العناصر التي عددها m، يوجد n عنصرًا فقط في اللائحة L، و $m - n$ عنصرًا في اللائحة الحرة. اكتب الإجراء COMPACTIFY-LIST(L, F) الذي يأخذ اللائحة L واللائحة الحرة F، وينقل عناصرَ L بحيث تَشْغَل هذه العناصر مواقع الصفيفة $1, 2, \ldots, n$، ويَضبط اللائحةَ الحرة F بحيث تبقى صحيحة، وبحيث تَشغل المواقع $n + 1, n + 2, \ldots, m$ في الصفيفة. يجب أن يكون زمن تنفيذ هذا الإجراء $\Theta(n)$، ويجب أن يَستخدم كميةً ثابتة من المساحة الإضافية فقط. ناقش بتأنٍّ صحة إجرائك.

4.10 تمثيل الأشجار ذوات الجذور

يمكن تطبيق طرائق تمثيل اللوائح التي رأيناها في المقطع السابق على أية بنية معطيات متجانسة. في هذا المقطع، ندرس على وجه التخصيص مسألةَ تمثيل الأشجار ذوات الجذور باستخدام بنى المعطيات المترابطة. نبدأ أولاً بدراسة الأشجار الثنائية ثم نقدِّم طريقةً للأشجار ذوات الجذور التي يكون للعقد فيها عددٌ اعتباطي من الأبناء.

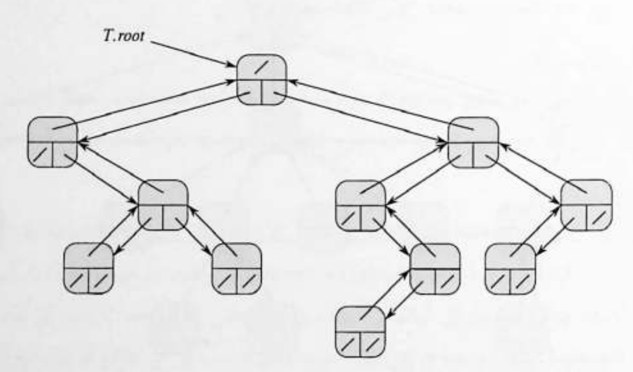
تمثِّل كلَّ عقدةٍ من الشجرة بغرض. وكما في اللوائح المترابطة، نفترض أن كلَّ عقدةٍ تحتوي الواصفة key. أما بقية الواصفات التي تهمنا، فهي مؤشرات إلى عقدٍ أخرى، وتختلف تبعًا لنوع الشجرة.

الأشجار الثنائية

يبيّن الشكل 9.10 كيف نستخدم الواصفات p و $left$ و $right$ لخزن مؤشرات إلى الأب، والابن الأيسر، والابن الأيمن، لكلِّ عقدةٍ في شجرة ثنائية T. إذا كان $x.p =$ NIL، فإن x يكون هو الجذر. وإذا لم يكن للعقدة x ابن أيسر، فإن $x.left =$ NIL، ومثل ذلك في حالة الابن الأيمن. يشار إلى جذر الشجرة T كلها بالواصفة $T.root$. فإذا كان $T.root =$ NIL فهذا يعني أن الشجرة فارغة.

الأشجار ذوات الجذور والتفرع غير المحدود

يمكن أن ينطبق منهج تمثيل الشجرة الثنائية على أيِّ صفٍّ من الأشجار يكون فيه عددُ أبناء كلِّ عقدة ثابتًا ما k على الأكثر: نستعيض عن الواصفات $left$ و $right$ بـ $child_1, child_2, \ldots, child_k$. لا يصلح هذا المنهج عندما يكون عدد أبناء العقدة غير محدود، مادمنا لا نعرف عدد الواصفات (الموافق لعدد الصفيفات



الشكل 9.10 تمثيل الشجرة الثنائية T. تمتلك كل عقدة x الواصفات $x.p$ (في الأعلى)، و $x.left$ (في الأدنى يسارًا)، و $x.right$ (في الأدنى يمينًا). لا تظهر هنا الواصفات key.

في التمثيل باستخدام عدة صفيفات) اللازم تخصيصه مقدَّمًا. أضف إلى ذلك، أنه إذا كان عدد الأبناء k محدودًا بثابت كبير، وكان لدى معظم العقد عددٌ صغير من الأبناء، فإننا سنخسر قدرًا كبيرًا من الذاكرة.

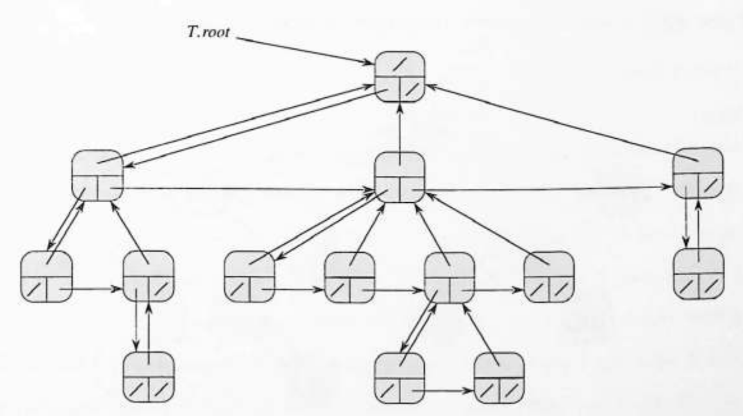
لحسن الحظ، يوجد منهج ذكي لتمثيل الأشجار ذوات العدد الاعتباطي من الأبناء. يتميز هذا المنهج باستخدام $O(n)$ فقط من مساحة التخزين لأية شجرة ذات جذر فيها n عقدة. يبيِّن الشكل 10.10 ***التمثيل باستخدام الابن الأيسر والشقيق الأيمن*** ***left child, right sibling representation***. وكما رأينا سابقًا، تحتوي كلُّ عقدة على مؤشر أبي p، و $T.root$ الذي يشير إلى جذر الشجرة T. وبدلاً من أن يكون لكل عقدة x مؤشر إلى كلِّ ابنٍ من أبنائها، يكون لها مؤشران فقط:

1. المؤشر $x.left\text{-}child$ الذي يشير إلى ابن العقدة x الموجود في أقصى اليسار.
2. المؤشر $x.right\text{-}sibling$ الذي يشير إلى شقيق x الموجود مباشرة إلى يمينه.

إذا لم يكن للعقدة x أبناء، يكون $x.left\text{-}child = \text{NIL}$، وإذا كانت العقدة x هي الابن الموجود في أقصى اليمين، يكون $x.right\text{-}sibling = \text{NIL}$.

طرق أخرى لتمثيل الأشجار

نلجأ أحيانًا إلى تمثيل الأشجار ذوات الجذور بطرق أخرى. ففي الفصل 6 مثلاً، مثّلنا الكومة - المعتمدةَ على شجرةٍ ثنائية كاملة - باستخدام صفيفةٍ وحيدة مع دليل العقدة الأخيرة في الكومة. وأما الأشجار التي تظهر في الفصل 21، فتعبر باتجاه الجذر فقط، ولذلك لا يوجد سوى المؤشرات الآباء؛ فلا وجود لمؤشرات إلى الأبناء. وثمة كثيرٌ من المناهج الممكنة الأخرى، يُحدِّد التطبيقُ أفضلها.



الشكل 10.10 تمثيل الشجرة T باستخدام الابن الأيسر والشقيق الأيمن. تمتلك كل عقدة x الواصفات $x.p$ (في الأعلى)، و $x.left\text{-}child$ (في الأدنى يسارًا) و $x.right\text{-}sibling$ (في الأدنى يمينًا). لا تظهر هنا الواصفات key.

تمارين

1-4.10

ارسم الشجرة الثنائية التي يقع جذرها عند الدليل 6 والممثلة بالواصفات التالية:

right	*left*	*key*	index
3	7	12	1
NIL	8	15	2
NIL	10	4	3
9	5	10	4
NIL	NIL	2	5
4	1	18	6
NIL	NIL	7	7
2	6	14	8
NIL	NIL	21	9
NIL	NIL	5	10

2-4.10

اكتب إجراءً عَوْديًّا زمنه $O(n)$ يطبع مفتاحَ كلِّ عقدةٍ في شجرة ثنائية معطاة ذات n عقدة.

3-4.10

اكتب إجراءً غير عَوْدي زمنه $O(n)$ يطبع مفتاحَ كلِّ عقدةٍ في شجرة ثنائية معطاة ذات n عقدة. استخدم مكدِّسًا باعتباره بنية معطيات مساعدة.

4-4.10

اكتب إجراءً زمنه $O(n)$ يطبع جميعَ المفاتيح في شجرة ذات جذر اعتباطية فيها n عقدة، علمًا بأن الشجرةَ خُزِّنت وفق التمثيل باستخدام الابن الأيسر والشقيق الأيمن.

★ ***5-4.10***

اكتب إجراءً غير عَوْدي زمنه $O(n)$ يطبع مفتاحَ كلِّ عقدة في شجرة ثنائية معطاة ذات n عقدة. لا تَستخدم أكثر من مساحة تخزين إضافية ثابتة خارج الشجرة نفسها، ولا تغيِّر الشجرة ولو مؤقتًا أثناء الإجراء.

★ ***6-4.10***

يحتاج التمثيل باستخدام الابن الأيسر والشقيق الأيمن لشجرةٍ ذات جذر اعتباطية إلى ثلاثة مؤشرات لكل عقدة: *left-child* و *right-sibling* و *parent*. يمكن الوصولُ من أية عقدةٍ إلى أبيها وتحديدُه في زمنٍ ثابت، ويمكن الوصولُ إلى أبنائها وتحديدُهم في زمنٍ خطي متعلق بعدد الأبناء. بيِّن كيف تَستخدم مؤشرَيْن فقط وقيمةً بوليانيةً واحدةً في كل عقدة بحيث يمكن الوصولُ إلى أبي عقدةٍ أو إلى جميع أبنائها وتحديدُهم في زمنٍ خطي متعلق بعدد الأبناء.

مسائل

10-1 مقارنات بين اللوائح

ما هو زمن التنفيذ المقارب في أسوأ الحالات، لكل عملية من عمليات المجموعات الديناميكية المذكورة في كلٍّ من أنواع اللوائح الأربعة المذكورة في الجدول التالي:

	غير مرتبة أحادية الترابط	مرتبة أحادية الترابط	غير مرتبة مضاعفة الترابط	مرتبة مضاعفة الترابط
SEARCH(L, k)				
INSERT(L, x)				
DELETE(L, x)				
SUCCESSOR(L, x)				
PREDECESSOR(L, x)				
MINIMUM(L)				
MAXIMUM(L)				

10-2 الكومات القابلة للدمج باستخدام اللوائح المترابطة

تدعم ***الكومة القابلة للدمج mergeable heap*** العمليات التالية: MAKE-HEAP (التي تنشئ كومةً قابلةً للدمج فارغة) و INSERT و MINIMUM و EXTRACT-MIN و UNION.[1]

بيِّن كيف تُنجَز الكومات القابلة للدمج باستخدام اللوائح المترابطة في كلٍّ من الحالات التالية. حاول أن تجعل كلَّ عمليةٍ فعّالةً قدر المستطاع. حلِّل زمنَ تنفيذ كلِّ عمليةٍ بدلالة حجم المجموعة (المجموعات) الديناميكية التي تعمل عليها.

أ. اللوائح مرتبة.

ب. اللوائح غير مرتبة.

ت. اللوائح غير مرتبة، والمجموعات الديناميكية المطلوب دمجها منفصلة.

10-3 البحث في لائحة متراصّة مرتبة

اهتمّ التمرين 10.3-4 بكيفية الاحتفاظ بلائحةٍ من n عنصرًا بحيث تكون متراصّة في المواقع الـ n الأولى في

---

[1] لـمّا كنا قد عرّفنا الكومة القابلة للدمج بحيث تدعم العمليتين MINIMUM و EXTRACT-MIN فيمكننا أن نسميها الكومة وفق الأصغر القابلة للدمج mergeable min-heap. وبالمقابل، لو كانت تدعم العمليتين MAXIMUM و EXTRACT-MAX لسميناها الكومة وفق الأكبر القابلة للدمج mergeable max-heap.

صفيفةٍ ما. نفترض هنا أن جميع المفاتيح متمايزة، وأن اللائحة المتراصة مرتبة أيضًا، أي إن $key[i] <$ $key[next[i]]$ لجميع قيم $i = 1, 2, \dots, n$، حيث $next[i] \neq$ NIL. وسنفترض أيضًا أن لدينا متحولاً L يحتوي دليل العنصر الأول في اللائحة. ضمن هذه الفرضيات، يطلب أن تبيِّن أن الخوارزمية ذات العشوائية المضافة التالية يمكن أن تُستخدم للبحث ضمن اللائحة في زمن متوقع $O(\sqrt{n})$.

```
COMPACT-LIST-SEARCH(L, n, k)
 1  i = L
 2  while i ≠ NIL and key[i] < k
 3      j = RANDOM(1, n)
 4      if key[i] < key[j] and key[j] ≤ k
 5          i = j
 6          if key[i] == k
 7              return i
 8      i = next[i]
 9  if i == NIL or key[i] > k
10      return NIL
11  else return i
```

لو تجاهلنا الأسطر 3-7 من الإجراء، نحصُل على خوارزميةٍ عادية للبحث في لائحة مترابطة مرتبة، حيث يشير الدليل i إلى جميع المواقع في اللائحة، كلٌّ بدوره. ينتهي البحث عندما "يسقط" الدليل i عن نهاية اللائحة أو عندما يصبح $key[i] \geq k$. في هذه الحالة، إذا كان $key[i] = k$ فمن الواضح أننا وجدنا مفتاحًا قيمته k. أما إذا أصبح $key[i] > k$، فعندها، لن نجد أبدًا مفتاحًا قيمته k، ومن ثَم يكون التوقف عن البحث هو العمل الصائب.

تحاول الأسطر 3-7 أن تقفز قُدُمًا نحو موضعٍ j اختير اختيارًا عشوائيًّا. تفيدنا هذه القفزة إذا كان $key[j]$ أكبر من $key[i]$ ولكنه ليس أكبر من k؛ في مثل هذه الحالة، يعين j موقعًا في اللائحة كان i سيبلغه أثناء عملية البحث العادية في اللائحة. ونظرًا لأن اللائحة متراصة، فإننا نعلم أن أي اختيار لـ j بين 1 و n سيعطينا دليلاً لغرضٍ ما في اللائحة وليس لفراغٍ في اللائحة الحرة.

وبدلاً من أن نحلِّل أداء COMPACT-LIST-SEARCH مباشرةً، فإننا سنحلِّل خوارزميةً مرتبطةً بها وهي: COMPACT-LIST-SEARCH′ التي تنفِّذ حلقتين مستقلتين. تأخذ هذه الخوارزمية موسطًا إضافيًّا t يحدِّد حدًّا أعلى لعدد تكرارات الحلقة الأولى.

```
COMPACT-LIST-SEARCH′(L, n, k, t)
 1  i = L
 2  for q = 1 to t
 3      j = RANDOM(1, n)
 4      if key[i] < key[j] and key[j] ≤ k
```

```
5          i = j
6          if key[i] == k
7              return i
8   while i ≠ NIL and key[i] < k
9       i = next[i]
10  if i == NIL or key[i] > k
11      return NIL
12  else return i
```

لمقارنة تنفيذ الخوارزميتين COMPACT-LIST-SEARCH(L,n,k) و COMPACT-LIST-SEARCH′(L,n,k,t) نفترض أن متتالية الأعداد الصحيحة التي تُنتج عن استدعاءات RANDOM$(1,n)$ هي نفسها في الخوارزميتين.

أ. افترض أن COMPACT-LIST-SEARCH(L,n,k) تحتاج t تكرارًا لحلقة **while** الواردة في الأسطر 2-8. علِّل كون COMPACT-LIST-SEARCH′(L,n,k,t) تعيد نفس النتيجة، وأن العدد الكلي للتكرارات في كلٍّ من حلقتي **for** و **while** ضمن COMPACT-LIST-SEARCH′ هو t على الأقل.

في استدعاء COMPACT-LIST-SEARCH′(L,n,k,t)، ليكن X_t المتحول العشوائي الذي يصف المسافة في اللائحة المترابطة (عبر سلسلة المؤشرات $next$) بين الموقع i والموقع الموافق للمفتاح k المراد الوصول إليه بعد t تكرارًا في حلقة **for** في الأسطر 2-7.

ب. برِّر أن الزمن المتوقع لتنفيذ COMPACT-LIST-SEARCH′(L,n,k,t) هو $O(t+\mathrm{E}[X_t])$.

ت. بيِّن أن $\mathrm{E}[X_t] \le \sum_{r=1}^{n}(1-r/n)^t$. (*تلميح*: استخدم المعادلة ت.25.)

ث. بيِّن أن $\sum_{r=0}^{n-1} r^t \le n^{t+1}/(t+1)$.

ج. برهن أن $\mathrm{E}[X_t] \le n/(t+1)$.

ح. بيِّن أن COMPACT-LIST-SEARCH′(L,n,k,t) تُنفَّذ في زمنٍ متوقع $O(t+n/t)$.

خ. استنتج أن COMPACT-LIST-SEARCH تنفَّذ في زمن متوقع $O(\sqrt{n})$.

د. لماذا نفترض أن جميع المفاتيح في COMPACT-LIST-SEARCH متمايزة فيما بينها؟ برِّر لماذا لا تساعد القفزات العشوائية في المقاربة بالضرورة عندما تحتوي اللائحة على قيمٍ مكرَّرة للمفاتيح.

ملاحظات الفصل

تُعَدُّ كُتب Aho و Hopcroft و Ullman [6] و knuth [209] مراجعَ ممتازةً لبنى المعطيات البدائية. تشمل كتبٌ أخرى بنى المعطيات الأساسية وتنجيزها في لغة برمجة محددة. نذكر أمثلةً على هذا النوع من الكتب

الجامعية Goodrich و Tamassia [147] و Main [241] و Shaffer [311] و Weiss [352, 353, 354].
يقدِّم Gonnet [145] معطياتٍ تجريبيةً تتعلق بأداء العديد من عمليات بنى المعطيات.

إن أصولَ المكدِّسات والأرتال باعتبارها بنى معطيات في علم الحاسوب غيرُ واضحة، لأن المفاهيم الموافقة لها في الرياضيات والممارسات الورقية للأعمال كانت موجودة قبل إدخال الحواسيب الرقمية. ينوِّه Knuth [209] العالِمَ A. M. Turing بتطويره المكدِّسات لربط المساقات الفرعية عام 1947.

ويبدو أن بنى المعطيات المعتمِدة على المؤشرات هي من اختراع الناس. فوفقًا لـ Knuth، يبدو أنه قد جرى استخدام المؤشرات في الحواسيب الأولى مع الذواكر الأسطوانة، وقد مثَّلت لغةُ A-1 التي طوَّرها G. M. Hopper في 1951 الصيغَ الجبرية باعتبارها أشجارًا ثنائية. ويعترف Knuth بفضل لغة IPL-II التي طوَّرها A. Newell و J. C. Shaw و H. A. Simon عام 1956 لاعترافها بأهمية استخدام المؤشرات والتشجيع على استخدامها. وقد تضمنت لغة IPL-III المطوَّرة في 1957 عمليات المكدِّس صراحة.

11 جداول التلبيد

تتطلب الكثير من التطبيقات مجموعةً ديناميكية من المعطيات تدعم العمليات المعجمية فقط: INSERT و SEARCH و DELETE. فمثلاً، يحتفظ مترجمٌ يترجم لغةَ برمجةٍ ما بجدول رموز، مفاتيحُ العناصر فيه هي متتالياتٌ محرفيةٌ اعتباطية تقابِل المعرَّفات في اللغة. إن جدول التلبيد هو بنية معطيات فعالة لتنجيز المعاجم. ومع أن البحث عن عنصرٍ ما في جدول تلبيد يمكن أن يستغرق عملياً زمناً مماثلاً لزمن البحث عن عنصرٍ في لائحة مترابطة ($\Theta(n)$ في أسوأ الحالات)، فإن أداء التلبيد يُعَدُّ جيدًا جدًّا. فبفرضيات معقولة، يصبح الزمن الوسطي للبحث عن عنصرٍ ما في جدول تلبيد هو $O(1)$.

يعمِّم جدولُ التلبيد المفهومَ الأبسط للصفيفة العادية. تَسْتَخْدِمُ العنونة المباشرةَ في الصفيفة العادية بفعالية قدرتنا على تفحُّص أيِّ موقعٍ في صفيفةٍ ما، في زمن $O(1)$. يناقش المقطع 1.11 العنونة المباشرة بتفصيل أكبر. ويمكننا الاستفادة من العنونة المباشرة عندما يكون متاحًا لنا تخصيص صفيفةٍ تحتوي على موضعٍ واحد لكل مفتاح ممكن.

حين يكون عددُ المفاتيح المخزَّنة فعليًّا صغيرًا بالنسبة إلى العدد الكلي للمفاتيح الممكنة، تصبح جداول التلبيد بديلاً فعالاً للعنونة المباشرة لصفيفةٍ ما، لأن جدول التلبيد يَستخدم عادةً صفيفةً يتناسب حجمها طردًا مع عدد المفاتيح المخزَّنة فعليًّا. عوضًا عن استخدام المفتاح مباشرةً باعتباره دليلَ صفيفة، يجري *حساب* دليل الصفيفة من المفتاح. يقدِّم المقطع 2.11 الأفكار الرئيسة مع التركيز على "السَّلْسَلة chaining" باعتبارها طريقةً لمعالجة "التصادمات collisions"، التي يقابِل فيها أكثرُ من مفتاحٍ دليلَ الصفيفة نفسَه. أما المقطع 3.11 فيصف كيف يمكننا حساب دلائل الصفيفة من المفاتيح، باستخدام دوال التلبيد. سنقدِّم عدة متغيرات للموضوع الأساسي ونحلِّلها. يلقي المقطع 4.11 نظرة على "العنونة المفتوحة open addressing" التي هي طريقة أخرى لمعالجة التصادمات. وخلاصة القول هي أن التلبيد تقانة فعالة جدًّا وعملية جدًّا: تتطلب العمليات المعجمية الأساسية وسطيًّا زمنًا $O(1)$ فقط. يشرح المقطع 5.11 كيف يستطيع "التلبيد التام perfect hashing" دعم عمليات البحث في *أسوأ الحالات* بزمن $O(1)$، حين تكون مجموعة المفاتيح المخزنة ساكنة (أي حين لا تتغير مجموعة المفاتيح المخزنة إطلاقًا بعد خزنها).

1.11 جداول العنوان المباشر

العنونة المباشرة تقانة بسيطة تعمل جيدًا حين تكون مجموعة المفاتيح الكلية U صغيرة على نحو معقول. افترض أن تطبيقًا ما، يحتاج إلى مجموعة ديناميكية لكلٍّ عنصرٍ منها مفتاحٌ مسحوب من المجموعة الكلية $U = \{0, 1, \dots, m-1\}$، حيث m ليست كبيرة جدًا. وسنفترض أنه لا يمتلك عنصران المفتاحَ نفسه.

لتمثيل المجموعة الديناميكية، نستخدم صفيفة، أو ***جدول عنوان مباشر direct-address table***، يرمز إليه بـ $T[0..m-1]$ ، ويوافق كل موقع أو ***شَقْب slot*** في هذا الجدول مفتاحًا من المجموعة الكلية U. يوضح الشكل 1.11 هذه المنهجية؛ حيث يؤشر الشَّقْب k إلى عنصر في المجموعة مفتاحه k. إذا كانت المجموعة لا تحتوي على عنصر مفتاحه k، عندها يكون $T[k] = \text{NIL}$.

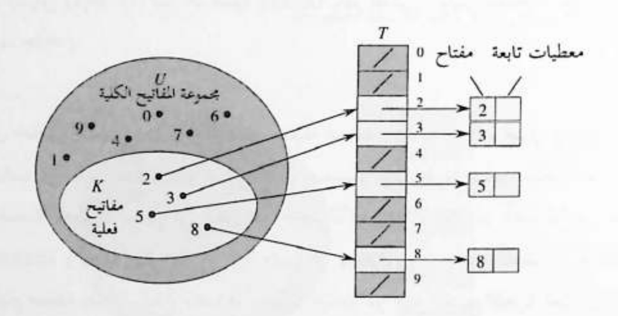
تُنجَزُ العمليات المعجمية ببساطة.

```
DIRECT-ADDRESS-SEARCH(T, k)
1  return T[k]

DIRECT-ADDRESS-INSERT(T, x)
1  T[x.key] = x

DIRECT-ADDRESS-DELETE(T, x)
1  T[x.key] = NIL
```

وتتطلب كلٌّ من هذه العمليات فقط زمنًا $O(1)$.



الشكل 1.11 تنجيز مجموعة ديناميكية باستخدام جدول عنوان مباشر T. يوافق كلُّ مفتاحٍ في المجموعة الكلية $U = \{0, 1, \dots, 9\}$ دليلاً في الجدول. تحدِّد مجموعة المفاتيح الفعلية $k = \{2, 3, 5, 8\}$ الشقوبَ التي تحتوي مؤشرات على عناصر في الجدول. الشقوب الأخرى، المظللة بالغامق تحتوي NIL.

في بعض التطبيقات، يمكن أن يحتوي جدولُ العنوان المباشر نفسه عناصرَ المجموعة الديناميكية. ذلك أنه عوضًا عن تخزين مفتاح عنصر ما والمعطيات التابعة له، في غرضٍ خارج جدول العنوان المباشر مع مؤشر من الشَّقْب في الجدول إلى الغرض، نستطيع تخزين الغرض نفسه في الشقب، وبذلك ندّخر فضاء ذاكرة. وقد نستخدم مفتاحًا خاصًا ضمن الغرض للإشارة إلى شَقْبٍ فارغ. يضاف إلى ذلك أنه ليس من الضروري، في غالب الأحيان، تخزين مفتاح الغرض، لأنه إذا توفر لدينا دليل الغرض في الجدول، أمكننا الحصول على مفتاحه. ولكن، إذا لم تكن المفاتيح مخزنة، وجب أن تتوفر طريقة تمكننا من معرفة كون الشقب فارغًا.

تمارين

1-1.11

افترض أن مجموعةً ديناميكيةً S ممثلةٌ بجدول عنوان مباشر T طوله m. صِفِ الإجراءَ الذي يكتشف العنصر الأعظمي في S. ما هو أداء إجرائك في أسوأ الحالات؟

2-1.11

شعاع البتات bit vector هو ببساطة صفيفة من البتات (أصفار ووحدان). يَشغل شعاع بتات طوله m فراغًا في الذاكرة أقل بكثير من صفيفة مكونة من m مؤشرًا. اشرح كيفية استخدام شعاع بتات لتمثيل مجموعة ديناميكية من العناصر المتمايزة دون معطيات تابعة لها. يجب أن تنفذ العمليات المعجمية في زمن $O(1)$.

3-1.11

اقترح طريقةً لتنجيز جدول عنوان مباشر لا تحتاج فيه مفاتيح العناصر المخزنة إلى أن تكون متمايزة، ويمكن للعناصر أن تحتوي معطيات تابعة لها. يجب أن تنفذ العمليات المعجمية الثلاث (INSERT و DELETE و SEARCH) في زمن $O(1)$. (لا تنسَ أن عملية DELETE تعتبر المؤشر – وليس المفتاح – هو الذي يُحدِّد الغرض المطلوب حذفه.)

★ 4-1.11

نرغب بتحقيق معجم باستخدام عنونة مباشرة على صفيفة ضخمة. في البداية، قد تحتوي عناصر الصفيفة فضلات معطيات، واستبداء الصفيفة غير عملي بسبب حجمها. صِفْ طريقةً لتنجيز معجم عنوان مباشر على صفيفة ضخمة. يجب أن يَشغل كلُّ غرض مخزَّن حجم ذاكرة $O(1)$؛ يجب أن تأخذ كلٌّ من عمليات INSERT و DELETE و SEARCH زمنًا $O(1)$؛ ويجب أن يستغرق استبداء بنية المعطيات زمنًا $O(1)$. (*تلميح*: استخدم صفيفة إضافية، تُعالَج باعتبارها مكدِّسًا حجمه هو عدد المفاتيح المخزنة فعليًّا في المعجم، وذلك للمساعدة في تحديد كون عُنصرٍ ما في الصفيفة الضخمة ما يزال مستخدمًا أم لا.)

2.11 جداول التلبيد

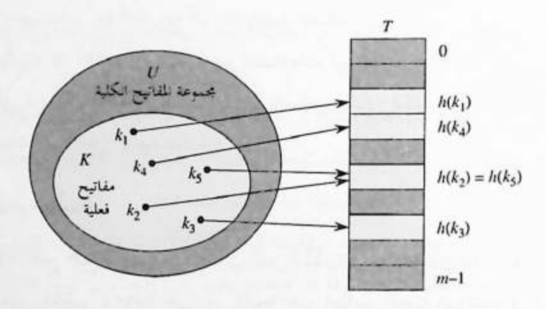
من المثالب الواضحة للعنونة المباشرة: إذا كانت المجموعة الكلية U كبيرة، فقد يكون من غير العملي، أو حتى المستحيل، تخزينُ جدولٍ T حجمُه $|U|$، إذا أخذنا بالحسبان حدود الذاكرة المتاحة في حاسوب عادي. إضافةً إلى أن مجموعة المفاتيح K *المخزّنة فعليًا* قد تكون صغيرةً جدًا مقارنةً بـ U، بحيث يضيع معظم الفراغ المحصص للجدول T.

إذا كانت مجموعة المفاتيح K المخزنة في المعجم أصغر بكثير من المجموعة الكلية U لجميع المفاتيح الممكنة، تطلّبَ جدول التلبيد ذاكرةً أقلَّ بكثير مما يتطلّبه جدول العنوان المباشر. وبعبارةٍ أدقّ، يمكننا أن نُخَفِّضَ متطلبات الذاكرة إلى حجم $\Theta(|k|)$ في حين نحافظ على فائدة أن يبقى زمن البحث عن عنصر ما في جدول تلبيد هو $O(1)$ فقط. إلا أن هذا الحدَّ يتعلق بالزمن *في الحالة الوسطى*، على حين ينطبق على *زمن البحث في أسوأ الحالات* في حال العنونة المباشرة.

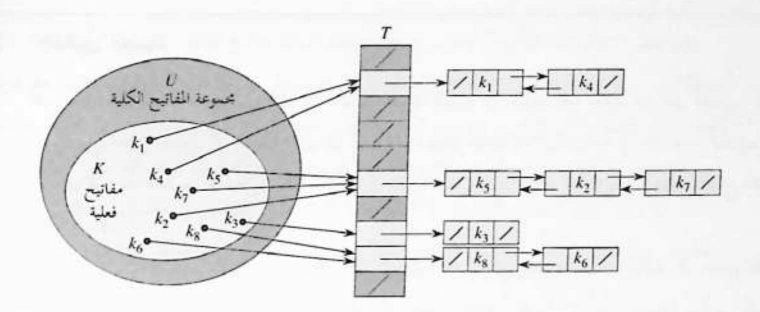
في العنونة المباشرة، يُخزَّن عنصر مفتاحه k في الشَّقْب k. أما في التلبيد، فيخزَّن هذا العنصر في الشقب $h(k)$ ؛ حيث، نستخدم ***دالة تلبيد*** h ***hash function*** لحساب شقب المفتاح k. هنا يقابل h بين مجموعة المفاتيح الكلية U وشقوب ***جدول التلبيد hash table*** $T[0\,..\,m-1]$:

$$h : U \rightarrow \{0, 1, \dots, m-1\} ,$$

حيث يكون حجمُ جدول التلبيد m عادةً أصغرَ بكثيرٍ من $|U|$. ونقول إن العنصر ذا المفتاح k ***يَتلبَّد hashes*** في الشقب $h(k)$؛ ونقول كذلك إن $h(k)$ هي ***قيمة تلبيد hash value*** المفتاح k. يوضِّح الشكل 2.11 الفكرة الرئيسة. إن دالة التلبيد تخفِّض مجال دلائل الصفيفة، ومن ثَمَّ حجم الصفيفة. فعوضًا عن الحجم $|U|$، قد تأخذ الصفيفة الحجم m.



الشكل 2.11 استخدام دالة تلبيد h لمقابلة مفاتيح بشقوب جدول تلبيد. ولما كان المفتاحان k_2 و k_5 يقابلان الشقب نفسه، فإنهما يتصادمان.



الشكل 3.11 حلّ التصادم بالسَّلْسَلَة. يحتوي كلُّ شقب $T[j]$ من جدول التلبيد لائحةً مترابطة مؤلَّفة من جميع المفاتيح التي قيمة تلبيدها j. مثلاً، $h(k_1) = h(k_4)$ و $h(k_5) = h(k_7) = h(k_2)$. يمكن أن تكون اللائحةُ المترابطةُ بسيطة الترابط أو مضاعفةَ الترابط؛ ونظهرها هنا على أنها مضاعفة الترابط لأن الحذف يكون أسرع في هذه الحالة.

ثمة عقبة واحدة: وهي أنه قد يتلبَّد مفتاحان في الشقب نفسه. نسمي هذه الحالة ***تصادمًا collision***. ولحسن الحظ، لدينا تقانات فعالة لإزالة التضارب الناتج عن التصادمات.

قد يكون الحل المثالي طبعًا هو في تجنب إحداث تصادمات. وقد نحاول تحقيق هذا الهدف باختيار دالة تلبيد مناسبة h. إحدى الأفكار المطروحة هي بجعل h تبدو "عشوائية"، وبذلك نتجنب التصادمات أو على الأقل نجعل عددها أصغريًا. يَستحضر مصطلح "يُلبِّد" صورًا من الخلط والتقطيع العشوائيين، تجعله يستحوذ على روح هذا النهج. (طبعًا، يجب أن تكون دالة التلبيد h حتمية بحيث ينتج دخل معطى k الخرج $h(k)$ نفسه دائمًا.) ولما كان $|U| > m$، وجب أن يمتلك مفتاحان على الأقل قيمة التلبيد نفسها؛ لذلك من المستحيل تجنب جميع التصادمات. وهكذا، ومع أن دالة التلبيد المصمَّمة جيدًا لتبدو "عشوائيةً" تخفِّف عدد التصادمات إلى الحد الأدنى، فإن الحاجةَ إلى طريقةٍ لتمييز التصادمات التي تَحدث تبقى قائمة.

يقدم القسم المتبقي من هذا المقطع أبسط تقانة لتمييز التصادم، تسمَّى السَّلسَلة. ويقدِّم المقطع 4.11 طريقة بديلة لتمييز التصادمات، تدعى العنونة المفتوحة.

تمييز التصادم باستخدام السَّلْسَلَة

في ***السَّلْسَلَة chaining***، نضع جميع العناصر التي تتلبَّد في نفس الشَّقْب في اللائحة المترابطة نفسها، كما يبين ذلك الشكل 3.11. يحتوي الشقب j مؤشرًا على رأس لائحة جميع العناصر المخزنة التي تتلبد في j؛ وإذا لم توجد مثل هذه العناصر، يحتوي الشقب j القيمة NIL.

يمكن تنجيز العمليات المعجمية على جدول التلبيد T بسهولة حين يجري تمييز التصادمات باستخدام السَّلْسَلَة.

```
CHAINED-HASH-INSERT(T, x)
1  insert x at the head of list T[h(x.key)]

CHAINED-HASH-SEARCH(T, x)
1  search for an element with key k in the list T[h(k)]

CHAINED-HASH-DELETE(T, x)
1  delete x from the list T[h(x.key)]
```

إن زمن تنفيذ عملية الإدراج في أسوأ الحالات هو $O(1)$. وإن إجراء الإدراج سريع إلى حدٍّ ما، لأنه يفترض أن العنصر x الذي هو في قيد الإدراج غير موجودٍ أصلاً في الجدول؛ ويمكننا أن نختبر هذا الافتراض عند الضرورة (بكلفة إضافية) بالبحث قبل الإدراج عن العنصر ذي المفتاح $x.key$. وأما زمن التنفيذ في أسوأ الحالات لعملية البحث، فيتناسب مع طول اللائحة؛ وسوف نحلِّل هذه العملية لاحقًا بعمق أكبر. يمكننا أن نحذف عنصرًا x في زمن $O(1)$ إذا كانت اللوائح مضاعفة الترابط، كما هو موضح في الشكل 3.11. (لاحظ أن CHAINED-HASH-DELETE يأخذ في الدخل عنصرًا x وليس مفتاحه k، لذا فإننا لا نحتاج إلى البحث عن x أولاً. فإذا كان جدول التلبيد يدعم الحذف، عندها يجب أن تكون لوائحُهُ المترابطة مضاعفةَ الترابط بحيث نستطيع حذف حدٍّ ما بسرعة. أما إذا كانت اللوائحُ بسيطةَ الترابط فقط - عند حذف عنصر ما x - وَجَبَ علينا أولاً أن نجده في اللائحة $T[h(x.key)]$ كي نَسْتَطيعَ تحديث الواصفة $next$ للعنصر السابق للعنصر x. هذا وتتساوى أزمنة التنفيذ المقاربة لكلٍّ من الحذف والبحث في اللوائح البسيطة الترابط.)

تحليل التلبيد مع السَّلْسَلَة

ما هي جودة أداء التلبيد مع السَّلْسَلَة؟ وعلى وجه الخصوص، كم يستغرق البحث عن عنصرٍ بدلالة مفتاحٍ معطى؟

ليكن لدينا جدولُ تلبيدٍ T مكونٌ من m شقبًا ويُخزِّن n عنصرًا، نعرِّف ***عامل التحميل*** ***load factor*** α لـ T بأنه n/m، وهو العدد الوسطي للعناصر المخزنة في سَلْسلة. سيكون تحليلنا بدلالة α، التي يمكن أن تكون أصغر من 1، أو مساوية له، أو أكبر منه.

إن سلوك التلبيد مع السَّلْسلة في أسوأ الحالات رديء جدًا، حيث تتلبد المفاتيح n كلُّها في الشقب نفسه، مشَكِّلَة لائحةً طولها n. وبذلك يكون زمن البحث في أسوأ الحالات $\Theta(n)$ إضافةً إلى زمن حساب دالة التلبيد — وهو ليس أفضل مما لو استخدمنا لائحةً واحدةً لكل العناصر. من الواضح أننا لا نستخدم

جداول التلبيد في حال أدائها في أسوأ الحالات. (يوفر التلبيد الكامل، المشروح في المقطع 5.11، أداءً جيدًا في أسوأ الحالات حين تكون مجموعة المفاتيح ساكنة.)

يعتمد أداء الحالة الوسطى للتلبيد على مدى الجودة التي تُوَزِّعُ بها دالة التلبيد h المفاتيح الواجب تخزينها بين m شَقْبًا، وسطيًّا. يناقش المقطع 3.11 هذه النقاط، لكننا سنفترض الآن أن أيَّ عنصرٍ معطى متساوي الأرجحية في أن يلبَّد في أيٍّ من الشُّقُوب m، بقطع النظر عن مكان تلبيد أي عنصرٍ آخر. نسمي هذا الافتراض ***التلبيد المنتظم البسيط*** ***simple uniform hashing***.

نرمز إلى طول اللائحة $T[j]$ بـ n_j، لكل $j = 0, 1, \ldots, m-1$. ومن ثَمّ يكون

$$n = n_0 + n_1 + \cdots + n_{m-1} , \qquad (1.11)$$

والتوقع لـ n_j هو $\mathrm{E}[n_j] = \alpha = n/m$.

نفترض أن الزمن $O(1)$ يكفي لحساب قيمة التلبيد $h(k)$، بحيث يتعلّق الزمن الذي يتطلبه البحث عن عنصرٍ مفتاحُهُ k خطيًّا بـ $n_{h(k)}$ طول اللائحة $T[h(k)]$. بإهمال الزمن $O(1)$ اللازم لحساب دالة التلبيد والنفاذ للشَّقْب $h(k)$، سندرس العدد المتوقع للعناصر التي تفحصها خوارزمية البحث، وهو عدد عناصر اللائحة $T[h(k)]$ التي تفحصها الخوارزمية لمعرفة أيٍّ من مفاتيحها يساوي k. سنأخذ بالحسبان حالتين؛ في الحالة الأولى لا ينجح البحث: أي لا يوجد في الجدول أيُّ عنصرٍ قيمةُ مفتاحه k. وفي الحالة الثانية ينجح البحث في إيجاد عنصرٍ مفتاحُهُ k.

مبرهنة 1.11

يستغرق البحثُ غيرُ الناجح في جدول تلبيدٍ جرى فيه حلّ التصادمات باستخدام السَّلْسَلة، زمنًا في الحالة الوسطى $\Theta(1+\alpha)$، وذلك بفرض أن التلبيد منتظم بسيط.

البرهان بافتراض أن التلبيد منتظم وبسيط، فإن أيَّ مفتاح k غير مُخَزَّن في الجدول يمكن أن يُلبَّدَ في أيٍّ من الشقوب m باحتمالٍ متساوٍ. وإن الزمن المتوقع للبحث غير الناجح لأي مفتاح k هو الزمن المتوقع للبحث حتى نهاية اللائحة $T[h(k)]$، ذات الطول المتوقع $\mathrm{E}[n_{h(k)}] = \alpha$. وبذلك، يكون العدد المتوقع للعناصر المفحوصة في بحث غير ناجح هو α، والزمن الكلي المطلوب (ومن ضمنه زمن حساب $h(k)$) هو $\Theta(1+\alpha)$. ■

يختلف الوضع قليلاً في حالة البحث الناجح ، إذ ليس لكلِّ لائحة الاحتمال نفسه بأن تكون عرضة للبحث. وبدلاً عن ذلك، يتناسب احتمال بحث اللائحة مع عدد العناصر التي تحتويها هذه اللائحة. ومع ذلك، يبقى الزمن المتوقع للبحث $\Theta(1+\alpha)$.

مبرهنة *2.11*

يستغرق البحث الناجح في جدول تلبيد جرى فيه تمييز التصادمات باستخدام السَّلْسَلة، زمنًا في الحالة الوسطى $\Theta(1+\alpha)$، وذلك بفرض أن التلبيد منتظم بسيط.

البرهان نفترض أن العنصر الذي نبحث عنه متساوي الاحتمال في أن يكونَ أيًا من العناصر n المخزَّنة في الجدول. يزيد عدد العناصر المختبرة خلال البحث الناجح عن عنصرٍ x بمقدار 1 على عدد العناصر التي تَظهر قبل x في لائحة x. ولما كانت العناصر الجديدة تُوضَعُ في مقدمة اللائحة، فإن العناصر الواقعة قبل x في اللائحة تكون قد أدرجت جميعُها بعد إدراج العنصر x. ولمعرفة العدد المتوقع للعناصر المفحوصة، سنأخذ المتوسط على n عنصرًا من نوع x في الجدول، وهو يساوي 1 مضافًا إليه العدد المتوقع للعناصر المضافة إلى لائحة x بعد إضافة x إلى اللائحة. نرمز بـ x_i إلى العنصر ذي الترتيب i الذي أدخل إلى الجدول، لكل $i = 1, 2, \ldots, n$، ولتكن $k_i = x_i.key$. نعرِّف متحولاً عشوائيًا مؤشرًا $X_{ij} = \mathrm{I}\{h(k_i) = h(k_j)\}$ لكلّ مفتاحين k_i و k_j. وبافتراض أن التلبيدَ منتظمٌ وبسيط، يكون لدينا $\Pr\{h(k_i) = h(k_j)\} = 1/m$، وباستخدام التوطئة 1.5 يكون لدينا كذلك $\mathrm{E}[X_{ij}] = 1/m$. وبهذا يكون العدد المتوقع للعناصر المفحوصة في البحث الناجح هو

$$
\begin{aligned}
&\mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}X_{ij}\right)\right] \\
&\quad = \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\mathrm{E}[X_{ij}]\right) && \text{(باستخدام خطية التوقع)} \\
&\quad = \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right) \\
&\quad = 1+\frac{1}{n\,m}\sum_{i=1}^{n}(n-i) \\
&\quad = 1+\frac{1}{n\,m}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right) \\
&\quad = 1+\frac{1}{n\,m}\left(n^2-\frac{n(n+1)}{2}\right) && \text{(باستخدام المعادلة (أ.1))} \\
&\quad = 1+\frac{n-1}{2m} \\
&\quad = 1+\frac{\alpha}{2}-\frac{\alpha}{2n}\,.
\end{aligned}
$$

وبذلك يكون الزمن الكلي المطلوب في حالة البحث الناجح (ومن ضمنه زمن حساب دالة التلبيد) $\Theta(2 + \alpha/2 - \alpha/2n) = \Theta(1 + \alpha)$. ■

ماذا يعني هذا التحليل؟ إذا كان عدد شقوب جدول التلبيد يتناسب على الأقل مع عدد العناصر في الجدول، يكون لدينا $n = O(m)$ ومن ثَم، $\alpha = n/m = O(m)/m = O(1)$. وبذلك، يستغرق البحث وسطيًا زمنًا ثابتًا. ولما كان الإدراج يستغرق زمنًا $O(1)$ في أسوأ الحالات، ويستغرق الحذف زمنًا $O(1)$ في أسوأ الحالات حينما تكون اللوائح مضاعفة الربط، فيمكننا أن ننفّذ جميع العمليات المعجمية في زمن $O(1)$ وسطيًا.

تمارين

1-2.11

لنفترض أننا نستخدم دالة تلبيد h لتلبيد n مفتاحًا متمايزًا في صفيفة T طولها m. وبافتراض أن التلبيد منتظمٌ بسيط، ما هو عدد التصادمات المتوقع؟ وبعبارةٍ أدقّ، ما هو عدد عناصر المجموعة $\{\{k, l\} : k \neq l \text{ and } h(k) = h(l)\}$؟

2-2.11

وضّح ما الذي يحدث عندما نُدرج المفاتيح 5, 28, 19, 15, 20, 33, 12, 17, 10 في جدول تلبيد جرى فيه تمييز التصادمات بالسَّلْسَلة. افترض أن الجدول يحتوي 9 شقوب، وأن دالة التلبيد هي $h(k) = k \bmod 9$.

3-2.11

يفترض البروفسور Marely أنه يمكنه الحصول على كسب جوهري في الأداء بوساطة تعديل طريقة السَّلْسَلة بحيث نحافظ على كل لائحة مرتبة ترتيبًا مفروزًا. كيف يؤثر تعديل البروفسور هذا على زمن التنفيذ في حالة البحث الناجح، والبحث غير الناجح، والإدراج، والحذف؟

4-2.11

اقترحْ طريقةً لتحصيص خزن العناصر وإزالة التحصيص ضمن جدول التلبيد نفسه، بربط جميع الشقوب غير المستخدمة ضمن لائحة الشقوب الفارغة. افترض أن الشَّقْبَ الواحد يمكن أن يُخَزِّن عَلَمًا flag إضافةً إلى عنصرٍ ومؤشرٍ أو مؤشرين. يجب أن تنفَّذ جميع العمليات المعجمية وعمليات لائحة الشقوب الفارغة في زمن متوقع $O(1)$. هل ينبغي أن تكون لائحةُ الشقوب الفارغة مضاعفةَ الترابط، أم يكفي أن تكون بسيطة الترابط؟

5-2.11

افترض أننا نُخَزِّنُ مجموعةً من n مفتاحًا في جدول تلبيد حجمه m. بَيّنْ أنه إذا كان يجري سحب المفاتيح من مجموعة U حيث $|U| > nm$، فإن هناك مجموعة جزئية من U حجمها n تتكون من المفاتيح التي تتلبد

جميعها في الشقب نفسه، بحيث يكون زمن البحث في أسوأ الحالات في حالة التلبيد مع السَّلْسَلة هو $\Theta(n)$.

6-2.11

افترض أننا خزّنّا n مفتاحًا في جدول تلبيد حجمه m، تُحَلُّ فيه التصادمات بوساطة السَّلْسَلة، وأننا نعلم طول كلّ سَلْسَلة، ومنها الطول L لأطول سَلْسَلة. اشرح الإجراء الذي يختار عشوائيًّا مفتاحًا من بين المفاتيح في جدول التلبيد وفق توزيع منتظم ويعيده في زمن متوقع $O(L \cdot (1 + 1/\alpha))$.

3.11 دوال التلبيد

سنناقش في هذا المقطع بعض النقاط التي تتعلق بتصميم دوال تلبيد جيدة، ثم نعرض ثلاثة مناهج لتعريف هذه الدوال. اثنان من هذه المناهج (وهما: التلبيد باستخدام التقسيم، والتلبيد باستخدام الضرب)، كسبيَّان heuristic بطبيعتهما، في حين يَستخدم المنهجُ الثالث (وهو التلبيد الشامل)، العشوائيةَ المضافة randomization ويتيح بفضلها أداءً جيدًا معزَّزًا بالبرهان.

ما الذي يجعل دالة التلبيد جيدة؟

تحقّق دالةُ التلبيد الجيدة (تقريبًا) افتراض التلبيد ذي التوزيع المنتظم البسيط[1]: أي إن لكلِّ مفتاحٍ احتمالاً متساويًا في أن يلبد إلى أيٍّ من الـ m شَقْبًا، باستقلالية عن مكان تلبيد أي مفتاح آخر. لسوء الحظ، لا يمكننا عادة التحقق من هذا الشرط، لأننا نادرًا ما نعرف التوزيع الاحتمالي الذي يجري سحب المفاتيح تبعًا له. يضاف إلى ذلك، أن سحْبَ المفاتيح قد لا يجري باستقلالية.

قد نعرف أحيانًا التوزيع. فمثلاً، إذا علمنا أن المفاتيح هي أعداد حقيقية عشوائية k مُوَزَّعَة باستقلالية وبانتظام في المجال $0 \le k < 1$، عندها تُحَقِّقُ دالة التلبيد

$h(k) = \lfloor km \rfloor$

شرط التلبيد المنتظم البسيط.

عمليًّا، يمكننا أن نستخدم غالبًا تقنياتٍ كسبيةً لإيجاد دالة تلبيد تؤدي أداءً جيدًا. وقد تفيد معلومات نوعية عن توزع المفاتيح في عملية التصميم. لنأخذ، مثلاً، جدول رموز في مُتَرْجِم compiler مفاتيحُهُ متتالياتٌ محرفية تمثّل مُعَرِّفات في برنامجٍ ما. غالبًا ما ترد رموزٌ وثيقة العلاقة، مثل `pt` و `pts`، في البرنامج نفسه. يُفترَض في دالة التلبيد الجيدة أن تقلِّل إلى الحد الأدنى فرصةَ تلبيد مثل هذه البدائل في الشقب نفسه.

تَشْتَقُّ منهجيةٌ جيدةٌ قيمةَ التلبيد بطريقةٍ نأمل أن تكون مستقلةً عن أية أنماط قد تكون موجودةً في

---

[1] للتبسيط، سندعو هذا التلبيد فيما يلي بالتلبيد المنتظم البسيط. (المراجع العلمي)

المعطيات. فمثلاً، تَحْسُبُ "طريقة التقسيم division method" (التي ناقشها في المقطع 1.3.11) قيمةَ التلبيد باعتبارها باقي قسمة المفتاح على عددٍ أوليٍّ محدَّد. تعطي هذه الطريقة غالبًا نتائج جيدة، بفرض أننا نختار العدد الأولي بحيث لا يكون متعلقًا بأي من الأنماط في توزع المفاتيح.

أخيرًا، نلاحظ أن بعض تطبيقات دوال التلبيد قد تتطلب خصائص أقوى مما هو متوفر في التلبيد المنتظم البسيط. مثلاً، قد نريد أن تعطي مفاتيحُ "متقاربة" بمعنًى ما قيمَ تلبيدٍ متباعدة فيما بينها. (هذه الخاصية مرغوبة خصوصًا عندما نستخدم السبر الخطي linear probing، المعرَّف في المقطع 4.11.) ويوفر التلبيد الشامل universal hashing، الموصوف في المقطع 3.3.11، الخاصيات المرغوبة.

تفسير المفاتيح كأعداد طبيعية

تفترض معظم دوال التلبيد أن المجموعة الشاملة للمفاتيح هي مجموعة الأعداد الطبيعية $\mathbb{N} = \{0, 1, 2, \dots\}$. فإذا لم تكن المفاتيح أعدادًا طبيعية، نوجد طريقة لتفسيرها كأعداد طبيعية. مثلاً، يمكننا تفسير المتوالية المحرفية كعددٍ صحيح يُعَبَّر عنه بتدوين باستخدام أساس مناسب. وهكذا، فقد نفسر المُعَرِّف `pt` كزوجٍ من الأعداد العشرية الصحيحة $(112, 116)$، لأن $p = 112$ و $t = 116$ في مجموعة محارف الـ ASCII؛ ثم يُعبَّر عن `pt` بوصفه عددًا صحيحًا حسب الأساس-128، فيصبح `pt` $(112 \cdot 128) + 116 = 14452$. نقوم عادةً في سياق تطبيقٍ ما بتصميم طريقةٍ كهذه لتفسير كلِّ مفتاح كعدد طبيعي (قد يكون عددًا كبيرًا). سنفترض فيما يلي أن المفاتيحَ أعدادٌ طبيعية.

1.3.11 طريقة التقسيم

لإنشاء دوال التلبيد باستخدام ***طريقة التقسيم division method***، نقابل مفتاحًا k بشَقْبٍ واحد من m شقبًا، وذلك بأخذ باقي قسمة k على m. بحيث تكون دالة التلبيد هي

$$h(k) = k \bmod m \ .$$

مثلاً، إذا كان حجم جدول التلبيد $m = 12$ والمفتاح $k = 100$، فإن $h(k) = 4$. ولما كانت طريقة التقسيم تحتاج إلى عمليةِ قسمةٍ واحدة، فإن التلبيدَ بالتقسيم سريعٌ جدًّا.

حين نَستخدم طريقةَ التقسيم، فإننا نتجنَّب عادة قيمًا معينة لـ m. مثلاً، يجب ألاَّ تكون m قوةً من قوى العدد 2، لأنه إذا كانت $m = 2^p$، فستكون الدالة $h(k)$ مجرَّد الـ p بتًّا ذات الأدنى مرتبةً من k. وما لم نكن نعلم أن جميعَ أنماط الـ p بتًّا الأدنى مرتبةً متساويةُ الاحتمال، فالأفضل أن نصمِّم دالةَ تلبيدٍ تعتمد على جميع بتات المفتاح. يُطلب إليك في التمرين 3.11-3 أن تُبيِّن أنَّ اختيار $m = 2^p - 1$، قد يكون خيارًا سيئًا، عندما تكون k متوالية محرفية مفسَّرة حسب قوى 2^p، لأن تبديل ترتيب محارف المفتاح k لا يغيِّر قيمة تلبيده.

يمثِّل عددٌ أولي غيرُ قريبٍ جدًّا من قوة للعدد 2 غالبًا خيارًا جيدًا لقيمة m. لنفترض مثلاً أننا نرغب

بتحصيص جدول تلبيد تُمايَزُ التصادمات فيه بالسَّلْسَلة، لتخزين $n = 2000$ متوالية محرفية على وجه التقريب، حيث يمثِّل المحرف بـ 8 بتات. ونحن لا نمانع اختبار 3 عناصر وسطيًّا في بحثٍ غير ناجح، لذا نخصِّص جدولَ تلبيد حجمه $m = 701$. اخترنا العدد 701 لأنه عددٌ أولي قريب من 2000/3 لكنَّهُ غير قريبٍ من أي قوة للعدد 2. بمعالجة أي مفتاح k كعدد صحيح، تكون دالة التلبيد

$$h(k) = k \bmod 701 \ .$$

2.3.11 طريقة الضرب

تعمل ***طريقة الضرب multiplication method*** على إيجاد دوال تلبيد على مرحلتين. في المرحلة الأولى نضرب المفتاح k بثابت A يقع ضمن المجال $0 < A < 1$ ونستخرج الجزء الكسري من kA. ثُمَّ نضرب هذه القيمة بـ m ونأخذ أقرب عدد صحيح للنتيجة. وباختصار، دالة التلبيد هي

$$h(k) = \lfloor m(kA \bmod 1) \rfloor \, ,$$

حيث تعني العبارة "$kA \bmod 1$" الجزءَ الكسري من kA، أي: $kA - \lfloor kA \rfloor$.

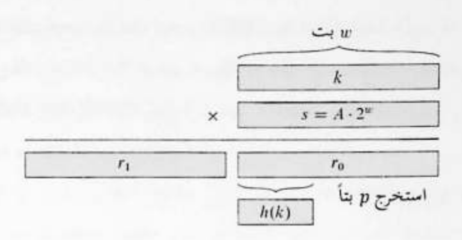
تكمن إحدى حسنات طريقة الضرب في أن قيمة m ليست ذات أهمية حرجة. ونختارها عادةً لتكون قوةً من قوى 2 ($m = 2^p$ و p عدد صحيح ما) لأننا نستطيع عندها تنجيز الدالة ببساطة على معظم الحواسيب كما يلي. افترض أن حجم كلمة الحاسوب هو w بتًّا، وأن k يتَّسِعُ في كلمة واحدة. نقيِّد قيم A لتكون كسرًا من الشكل $s/2^w$، حيث s عدد صحيح في المجال $0 < s < 2^w$. بالإشارة إلى الشكل 4.11، نضرب أولاً k بالعدد الصحيح ذي الـ w-بتًّا وهو $s = A \cdot 2^w$، فتكون النتيجة هي القيمة $r_1 2^w + r_0$ التي عرضها $2w$ -بتًّا، حيث r_1 هي كلمة الجداء الأعلى مرتبة، و r_0 هي كلمة الجداء الأدنى مرتبة. وتتألف قيمة التلبيد p-بتًّا المطلوبة من p-بتًّا الأكثر دلالة من r_0.

ومع أن هذه الطريقة تعمل على أيِّ قيمةٍ للثابت A، إلا أنها تعمل على نحو أفضل على قيمٍ دون أخرى. ويَعتمد الخيار الأمثل على خصائص المعطيات المُلَبَّدَة. يقترح Knuth [211] أنه من المرجَّح أن تعمل القيمة

$$A \approx (\sqrt{5} - 1)/2 = 0.6180339887 \ldots \qquad (2.11)$$

بجودة معقولة.

لنفترض، كمثال على ذلك، أن $k = 123456$، و $p = 14$، و $m = 2^{14} = 16384$، و $w = 32$. وبأخذ اقتراح Knuth، نختار A كسرًا من الشكل $s/2^{32}$ بحيث يكون الأقرب إلى $(\sqrt{5} - 1)/2$، وبحيث $A = 2654435769/2^{32}$. إذن $k \cdot s = 327706022297664 = (2^{32} \cdot 76300) + 17612864$، وبذلك يكون $r_1 = 76300$ و $r_0 = 17612864$. وتُنْتِج الـ 14 بتًّا العليا من r_0 القيمة $h(k) = 67$.



الشكل 4.11 طريقة التلبيد بالضرب. يجري ضرب تمثيل المفتاح k بـ w-بتًّا بالقيمة $s = A \cdot 2^w$. تشكّل الـ p بتًّا من المرتبة الأعلى من النصف الأدنى ذي العرض w -بتًّا من الجداء قيمة التلبيد المطلوبة $h(k)$.

★ 3.3.11 التلبيد الشامل

إذا اختار خصم ماكر أن تُلَبّدَ المفاتيح باستخدام دالة تلبيد ثابتة، عندها يستطيع اختيار n مفتاحًا تَتَلَبّدُ جميعها في الشَّقْب نفسه، وينتج متوسط زمن استرجاع $\Theta(n)$. إن أية دالة تلبيد ثابتة معرضة لمثل هذا السلوك الرديء في أسوأ الحالات؛ الطريقة الفعالة الوحيدة لتحسين هذا الوضع هي باختيار دالة التلبيد *عشوائيًا* بطريقة *مستقلة* عن المفاتيح التي ستُخَزَّنُ فعليًا. يمكن أن تؤدي هذه المنهجية، التي تسمى ***التلبيد الشامل universal hashing***، بالمتوسط إلى أداء جيد يمكن إثباته، بقطع النظر عن المفاتيح التي يختارها الخصم.

في التلبيد الشامل، نختار في بداية التنفيذ دالة التلبيد عشوائيًا من مجموعة دوال مصممة. حيث تضمن إضافة العشوائية، كما هو الحال في الفرز السريع، ألا يؤدي دخل وحيد دائمًا إلى سلوك أسوأ الحالات. وبسبب أننا نختار دالة التلبيد عشوائيًا، تستطيع الخوارزمية أن تسلك سلوكًا مختلفًا في كل تنفيذ، حتى في حالة الدخل نفسه، ضامنة أداءً جيدًا في الحالة الوسطى لأي دخل. وبالعودة إلى مثال جدول رموز المترجم، نجد أن اختيار المبرمج للمُعَرِّفات لا يمكن أن يسبب الآن أداءَ تلبيدٍ سيئًا ثابتًا. ويَحدث الأداءُ السَّيئُ فقط عندما يختار المترجم دالة تلبيد عشوائية تؤدي إلى تلبيد مجموعة من المُعَرِّفات برداءة، لكن احتمال حدوث هذا الوضع يبقى صغيرًا، ويحدث الأمر نفسه لأي مجموعة مُعَرِّفاتٌ لها الحجم نفسه.

لتكن $\mathcal{H}$ مجموعة منتهية من دوال التلبيد التي تقابل عالماً معطى من المفاتيح U بقيم من المجموعة $\{0, 1, \ldots, m-1\}$. يقال عن مثل هذه المجموعة أنها ***شاملة universal*** إذا تحقق ما يلي: لكل زوج من المفاتيح المتمايزة $k, l \in U$، يكون عدد دوال التلبيد $h \in \mathcal{H}$ حيث $k(k) = h(l)$ هو على الأكثر $|\mathcal{H}|/m$. وبعبارة أخرى، باختيار دالة تلبيد عشوائيًا من $\mathcal{H}$، فإن احتمال التصادم بين مفتاحين متمايزين k و l لا يزيد على احتمال $1/m$ من التصادم إذا اختير $h(k)$ و $h(l)$ عشوائيًا وعلى نحو مستقل من المجموعة $\{0, 1, \ldots, m-1\}$.

تبيِّن المبرهنة التالية أن سلوك صف دوال التلبيد الشامل سلوكٌ جيد في الحالة الوسطى. تذكَّر أن n_i تشير إلى طول اللائحة $T[i]$.

مبرهنة 3.11

افترض أنه يجري اختيار دالة تلبيد h عشوائيًا من مجموعة شاملة من دوال التلبيد واستخدمناها لتلبيد n مفتاحًا في جدول T حجمه m، واستخدمنا السَّلْسَلة لتمييز التصادم. إذا لم يكن المفتاح k في الجدول، عندها يكون الطول المتوقع $\mathrm{E}[n_{h(k)}]$ للائحة التي يتلبد فيها المفتاح k هو على الأكثر عامل التحميل $\alpha = n/m$. وإذا وُجد المفتاح k في الجدول، عندها يكون الطول المتوقع $\mathrm{E}[n_{h(k)}]$ للائحة التي تحتوي المفتاح k هو $\alpha + 1$ على الأكثر.

البرهان نلاحظ هنا أن التوقعات هي على اختيار دالة التلبيد، ولا تعتمد على أية فرضيات تتعلق بتوزع المفاتيح. نعرّف لكل زوج من المفاتيح المتمايزة k و l، متحولاً عشوائيًا مؤشرًا $X_{kl} = \mathrm{I}\{h(k) = h(l)\}$. ولأنه حسب تعريف مجموعة شاملة من دوال التلبيد، يتصادم زوج وحيد من المفاتيح باحتمال $1/m$ على الأكثر، فلدينا $\Pr[h(k) = h(l)] \le 1/m$، وبذلك تقتضي التوطئة 1.5 أن يكون $\mathrm{E}[X_{kl}] \le 1/m$.

بعد ذلك، نُعَرِّفُ، لكل مفتاح k، المتحول العشوائي Y_k الذي يساوي عدد المفاتيح غير المفتاح k التي تتلبد في الشَّقْب نفسه الذي يتلبد فيه المفتاح k، بحيث

$$Y_k = \sum_{\substack{l \in T \\ l \ne k}} X_{kl} \ .$$

وبذلك يكون لدينا

$$\begin{aligned} \mathrm{E}[Y_k] &= \mathrm{E}\left[\sum_{\substack{l \in T \\ l \ne k}} X_{kl}\right] \\ &= \sum_{\substack{l \in T \\ l \ne k}} \mathrm{E}[X_{kl}] \qquad \text{(باستخدام خطية التوقع)} \\ &\le \sum_{\substack{l \in T \\ l \ne k}} \frac{1}{m} \ . \end{aligned}$$

تعتمد تتمة البرهان على كون المفتاح k في الجدول T أو لا.

- إذا كان $k \notin T$، فإن $n_{h(k)} = Y_k$ و $|\{l : l \in T \text{ and } l \ne k\}| = n$. ومن ثَم يكون $\mathrm{E}[n_{h(k)}] = \mathrm{E}[Y_k] \le n/m = \alpha$.

• إذا كان $k \in T$، ولما كان المفتاح k يظهر في اللائحة $T[h(k)]$ والعدد Y_k لا يشمل المفتاح k، يكون لدينا $n_{h(k)} = Y_k + 1$ و $|\{l : l \in T \text{ and } l \neq k\}| = n - 1$. وهكذا يكون $\mathrm{E}[n_{h(k)}] = \mathrm{E}[Y_k] + 1 \leq (n-1)/m + 1 = 1 + \alpha - 1/m < 1 + \alpha$. ■

النتيجة التالية تقول أن التلبيد الشامل يوفر الربح المطلوب: فالآن يصبح من المستحيل أن يجد خصمٌ متتاليةً من العمليات تُجبر زمن التنفيذ على أن يكون في أسوأ حالاته. وبإضافة عشوائية على اختيار دالة التلبيد أثناء التنفيذ، على نحو ذكي، نضمن أن نستطيع معالجة أية متتالية من العمليات، في زمن تنفيذ جيد في الحالة المتوسطة.

نتيجة 4.11

باستخدام التلبيد الشامل مع تمييز التصادم بالسَّلْسَلة في جدول فارغ في البداية ويحتوي m شَقْبًا، تستغرق معالجة أية متتالية، مؤلَّفةٍ من n عملية INSERT و SEARCH و DELETE وتحتوي $O(m)$ عملية INSERT، زمنًا متوقعًا $\Theta(n)$.

البرهان لما كان عدد عمليات الإدراج $O(m)$، فلدينا $n = O(m)$ وكذلك $\alpha = O(1)$. وتستغرق عمليتا INSERT و DELETE زمنًا ثابتًا، وحسب المبرهنة 3.11، فإن الزمن المتوقع لكل عملية SEARCH هو $O(1)$. وباستخدام خاصية خطية التوقع ينتج أن الزمن المتوقع لمتتالية العمليات n كلها هو $O(n)$. ولما كانت كل عملية تستغرق زمنًا $\Omega(1)$، فينتج لدينا الحد $\Theta(n)$. ■

تصميم صف شامل من دوال التلبيد

من السهل جدًّا تصميم صفٍّ شامل من دوال التلبيد، لأن نظرية الأعداد الصغيرة تساعدنا على إثبات ذلك. قد تحتاج أولاً للرجوع إلى الفصل 31 إذا لم تكن نظرية الأعداد مألوفة لك.

نبدأ باختيار عدد أولي p كبير كفاية بحيث يقع كل مفتاح k في المجال 0 إلى $p - 1$ ، ضمنًا. نرمز بـ $\mathbb{Z}_p$ إلى المجموعة $\{0, 1, \ldots, p-1\}$ وبـ $\mathbb{Z}_p^*$ إلى المجموعة $\{1, 2, \ldots, p-1\}$. ولما كان p عددًا أوليًّا، فإننا نستطيع حل المعادلات بالمقاس p باستخدام الطرق المعطاة في الفصل 31. ولأننا نفترض حجم المجموعة الشاملة للمفاتيح أكبر من عدد شقوب جدول التلبيد، يكون لدينا $p > m$.

نعرِّف الآن دالة التلبيد h_{ab} لأي $a \in \mathbb{Z}_p^*$ وأي $b \in \mathbb{Z}_p$ باستخدام التحويل الخطي متبوعًا باختزالات المقاس p ، ثم المقاس m:

$$h_{ab}(k) = ((ak + b) \bmod p) \bmod m \ . \qquad (3.11)$$

فمثلاً، في حال $p = 17$ و $m = 6$، لدينا $h_{3,4}(8) = 5$. وتكون جماعة كل دوال التلبيد المماثلة هي

$$\mathcal{H}_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\} \ . \qquad (4.11)$$

تطابق كلُّ دالة تلبيد h_{ab} المجموعةَ $\mathbb{Z}_p$ إلى $\mathbb{Z}_m$. ولهذا الصف من دوال التلبيد خاصية دقيقة وهي أن حجم مجال الخرج m اعتباطي — ليس بالضرورة عددًا أوليًا — وهي سمة سنستخدمها في المقطع 5.11. ولما كان لدينا $p-1$ خيارًا لـ a، و p خيارًا لـ b، فالمجموعة $\mathcal{H}_{pm}$ ستحوي $p(p-1)$ دالة تلبيد.

مبرهنة 5.11

إن الصف $\mathcal{H}_{pm}$ من دوال التلبيد المعرَّف بالمعادلتين (3.11) و (4.11) هو شامل.

البرهان نأخذ مفتاحين متمايزين k و l من $\mathbb{Z}_p$، حيث $k \neq l$. وليكن، في حالة دالة تلبيد معطاة h_{ab}،

$$r = (ak + b) \bmod p \ ,$$
$$s = (al + b) \bmod p \ .$$

نلاحظ أولاً أن $r \neq s$. لماذا؟ لاحظ أن

$$r - s \equiv a(k - l) \pmod p \ .$$

وهذا يستتبع أن $r \neq s$ لأن p أولي وكلاً من a و $(k-l)$ لا يساوي الصفر بالمقاس p، إذن يجب أن يكون جداؤهما لا يساوي الصفر بالمقاس p حسب المبرهنة 6.31. لذلك، عند حساب أي h_{ab} إشارة انتماء $\mathcal{H}_{pm}$، تقابَل المدخلات المتمايزة k و l بقيم r و s متمايزة بالمقاس p؛ ولن يوجد تصادم على "مستوى المقاس p". إضافة إلى ذلك، يؤدي كل من الخيارات $p(p-1)$ للزوج (a,b) وحيث $a \neq 0$ إلى زوج ناتج ***مختلف*** (r,s) حيث $r \neq s$، لأننا نستطيع حل a و b إذا أُعطينا r و s:

$$a = \left((r - s)((k - l)^{-1} \bmod p)\right) \bmod p \ ,$$
$$b = (r - ak) \bmod p \ ,$$

حيث يشير $((k-l)^{-1} \bmod p)$ إلى المقلوب الجدائي[2]، الوحيد، بالمقاس p للمقدار $(k-l)$. ولما كان لدينا $p(p-1)$ زوجًا ممكنًا (r,s) فقط حيث $r \neq s$، فهنالك تقابل واحد–إلى–واحد بين الأزواج (a,b) حيث $a \neq 0$ والأزواج (r,s) حيث $r \neq s$. وبذلك، وفي حال أي زوج من المدخلات المعطاة k و l، إذا انتقينا (a,b) على نحو عشوائي بانتظام من $\mathbb{Z}_p^* \times \mathbb{Z}_p$، فسيكون للزوج الناتج (r,s) احتمال متساوٍ في أن يكون أي زوج من القيم المتمايزة بالمقاس p.

ينتج من ذلك أن احتمال تصادم المفتاحين المتمايزين k و l يساوي احتمال $r \equiv s \pmod m$ عند اختيار r و s عشوائيًا باعتبارهما قيمًا متمايزة بالمقاس p. في حال قيمة معطاة r، وللقيم الممكنة المتبقية لـ s، وعددها $(p-1)$، فإن عدد قيم s حيث $s \neq r$ و $s \equiv r \pmod m$ هي على الأكثر

[2] المقلوب الجدائي بالمقاس p لعدد ما m هو العدد الطبيعي الذي يكون ناتج جدائه بـ m بالمقاس p هو 1. مثال: 4 هو المقلوب الجدائي بالمقاس 11 للعدد 3 لأن $4 \times 3 \bmod 11 = 1$. (المراجع العلمي)

$\lceil p/m \rceil - 1 \le ((p+m-1)/m) - 1$ (حسب المتراجحة (6.3))

$= (p-1)/m$.

إن احتمال تصادم s مع r في حال الاختزال بالمقاس m هو على الأكثر

$((p-1)/m)/(p-1) = 1/m$.

وهكذا، في حال أي زوج من القيم المتمايزة $k, l \in \mathbb{Z}_p$،

$\Pr\{h_{ab}(k) = h_{ab}(l)\} \le 1/m$,

بحيث يكون $\mathcal{H}_{pm}$ شاملاً بالفعل. ■

تمارين

1-3.11

لنفترض أننا نريد البحث في لائحة مترابطة طولها n، حيث يحتوي كلُّ عنصرٍ مفتاحًا k وقيمةَ تلبيد $h(k)$. المفتاح k هو متتالية محرفية طويلة. كيف يمكننا الاستفادة من قيم التلبيد عند البحث في اللائحة عن عنصر له مفتاحٌ معطى؟

2-3.11

افترض أننا نلبّد متتاليةً من r محرفًا في m شَقْبًا بمعالجتها كعددٍ بالأساس-128، ثم باستخدام طريقة التقسيم. نستطيع بسهولة أن نُمثّلَ العدد m ككلمة حاسوب 32-بتًا، فيما تأخذ المتتاليةُ المؤلَّفة من r محرفًا، والمعالجَةُ كعددٍ بالأساس-128، عدةَ كلمات. كيف يمكننا تطبيق طريقة التقسيم لحساب قيمة تلبيد متتاليةٍ محرفيةٍ دون أن نستخدم أكثرَ من عددٍ ثابت من كلمات الذاكرة خارج المتتالية نفسها؟

3-3.11

لتكن لدينا نسخة من طريقة التقسيم فيها دالة التلبيد $h(k) = k \bmod m$، حيث $m = 2^p - 1$ و k هو متتالية محرفية تفسَّر في الأساس 2^p. بيِّن أننا لو استطعنا اشتقاق المتتالية x من المتتالية y بتبديل محارفها، فإن x و y تتلبَّدان إلى القيمة نفسها. أعطِ مثالاً من تطبيقٍ تكون فيه هذه الخاصية غير مرغوبة في دالة التلبيد.

4-3.11

ليكن لدينا جدول تلبيد حجمه $m = 1000$ ودالة التلبيد الموافقة هي $h(k) = \lfloor m(kA \bmod 1) \rfloor$ حيث $A = (\sqrt{5}-1)/2$. احسب المواضعَ التي تنطابق إليها المفاتيح 61 و 62 و 63 و 64 و 65.

5-3.11 ★

عرف جماعةً من دوال تلبيد $\mathcal{H}$ من مجموعةٍ منتهية U في مجموعةٍ منتهية B لتكون ***ϵ-universal*** إذا تحقق لكل الأزواج من العناصر المتمايزة k و l في U،

$\Pr\{h(k) = h(l)\} \le \epsilon$,

حيث يؤخذ احتمال سحب دالة التلبيد h من الجماعة $\mathcal{H}$ عشوائيًا. بيّن أن عائلة **ϵ-universal** من دوال التلبيد يجب أن تحقق

$$\epsilon \geq \frac{1}{|B|} - \frac{1}{|U|} .$$

★ *6-3.11*

لتكن U مجموعةً من القيم ذات n مركبة المسحوبة من $\mathbb{Z}_p$، ولتكن $B = \mathbb{Z}_p$، حيث p عددٌ أولي. عرّف دالة تلبيد $h_b : U \to B$ حيث $b \in \mathbb{Z}_p$، على دخلٍ ذي n مركبة $\langle a_0, a_1, \dots, a_{n-1} \rangle$ من U بحيث

$$h_b(\langle a_0, a_1, \dots, a_{n-1} \rangle) = \left(\sum_{j=0}^{n-1} a_j b^j \right) \bmod p \ ,$$

ولتكن $\mathcal{H} = \{h_b : b \in \mathbb{Z}_p\}$. ناقش كون $\mathcal{H}$ هي $((n-1)/p)$ –شاملة طبقًا لتعريف **ϵ-universal** الوارد في التمرين 11.3-5. (*تلميح:* انظر التمرين 11.4-4.)

4.11 العنونة المفتوحة

في *العنونة المفتوحة* ***open addressing***، تَشغل جميعُ العناصر جدولَ التلبيد نفسه؛ حيث يحتوي كلُّ عنصر لجدول التلبيد إما عنصرًا من المجموعة الديناميكية وإما NIL. في حال البحث عن عنصرٍ ما، نفحص شقوب الجدول بانتظام حتى نجد العنصر المطلوب أو نتحقَّق من عدم وجود العنصر في الجدول. وعلى عكس السَّلْسَلَة، لا توجد لوائح ولا عناصر مخزنة خارج الجدول. وهكذا، في العنونة المفتوحة، يمكن أن "يمتلئ" جدول التلبيد بحيث لا يمكن تنفيذ عمليات إدراجٍ إضافية؛ وبالنتيجة لا يمكن أن يتجاوز معامل التحميل α القيمة 1.

لا شكَّ في أنه كان باستطاعتنا تخزينُ لوائحَ مترابطة – لِسَلْسَلتها داخلَ الجدول – في شقوب الجدول غير المستخدَمة (انظر التمرين 11.2-4)، غير أن الفائدة من العنونة المفتوحة هي أنها تتجنب المؤشرات تمامًا. وعوضًا عن تتبع المؤشرات، نحسب تتالي الشقوب الواجب فحصها. وتوفر الذاكرة الإضافية المحرَّرة بسبب عدم تخزين المؤشرات عددًا أكبر من الشقوب، لحجم الذاكرة نفسه، مؤدِّيةً إلى تصادماتٍ أقلّ واسترجاعٍ أسرع.

لتنجيز الإدراج باستخدام العنونة المفتوحة، نفحص على التتابع، أو *نسبر* ***probe***، جدولَ التلبيد حتى نجد شَقْبًا فارغًا نضع فيه المفتاح. وعوضًا عن أن نكون مقيَّدين بالترتيب $0, 1, \dots, m-1$ (الذي يحتاج إلى زمن بحث $\Theta(n)$)، تعتمد متتالية المواقع التي تُسْبَرُ على المفتاح الذي نُدْرِجه. ولتحديد الشقوب التي ينبغي سبرها، نوسِّع دالة التلبيد لتشمل عدد السبر (ابتداءً من 0) باعتباره دخلاً ثانيًا. وهكذا، تصبح دالة التلبيد

$$h : U \times \{0, 1, \dots, m-1\} \to \{0, 1, \dots, m-1\} .$$

في العنونة المفتوحة، نطلب لكل مفتاحٍ k، أن تكون ***متتالية السبر probe sequence***

$\langle h(k,0), h(k,1), \dots, h(k,m-1) \rangle$

تبديلاً من $\langle 0,1,\dots,m-1 \rangle$، بحيث يُعتبر كل موقع من جدول التلبيد شَقْبًا لمفتاح جديد فيما يمتلئ الجدول. سنفترض في شبه الرماز التالي أن العناصر في جدول التلبيد T هي مفاتيحُ بدون معلومات تابعة؛ وأن المفتاح k متطابقٌ مع العنصر الذي يحتوي المفتاح k. إن كلَّ شَقْبٍ يحوي إما مفتاحًا أو NIL (إذا كان الشقب فارغًا). ويأخذ الإجراء HASH-INSERT في مدخلاته جدولَ تلبيد T ومفتاحًا k. ويعيد رقمَ الشقب حيث يُخَزَّنُ المفتاحَ k أو أن يُظْهِر خطأً لأن جدول التلبيد ممتلئ حاليًا.

```
HASH-INSERT(T, k)
1  i = 0
2  repeat
3      j = h(k, i)
4      if T[j] == NIL
5          T[j] = k
6          return j
7      else i = i + 1
8  until i == m
9  error "hash table overflow"
```

تسبر خوارزمية البحث عن مفتاح k متتالية الشقوب نفسها التي فحصتها خوارزمية الإدراج عند إدراج المفتاح k. لذا يمكن أن يتوقف البحث (مخفقًا) عندما يجد شقبًا فارغًا، لأن k كان سيُدرج في هذا الشقب وليس بعده حسب متتالية سبره. (يفترض هذا النقاش أن المفاتيح لا تحذف من جدول التلبيد.) يأخذ الإجراء HASH-SEARCH في مدخلاته جدول التلبيد T ومفتاحًا k، ويعيد j إذا كان الشقب j يحتوي المفتاح k، أو NIL إذا لم يكن المفتاح k موجودًا في الجدول T.

```
HASH-SEARCH(T, k)
1  i = 0
2  repeat
3      j = h(k, i)
4      if T[j] == k
5          return j
6      i = i + 1
7  until T[j] == NIL or i == m
8  return NIL
```

إن الحذف من جدول تلبيد بعنونة مفتوحة صعب. فعندما نحذف مفتاحًا k من شَقْب i، لا نستطيع ببساطة أن نُعَلِّمَ ذلك الشقب على أنه شقب فارغ بأن نَخْزِنَ فيه NIL. لأننا إذا فعلنا ذلك فقد يكون من المستحيل استعادة أي مفتاح k كنا قد سبرنا خلال إدراجه الشقبَ i ووجدناه مشغولاً. نستطيع أن نحل هذه

المسألة بأن نجعلَ الشقب يُخزِّنُ القيمةَ الخاصة DELETED عوضًا عن القيمة NIL. في هذه الحالة، علينا تعديل الإجراء HASH-INSERT لمعالجة هذا الشقب كما لو أنه كان فارغًا بحيث نستطيع أن نُدْرِج فيه مفتاحًا جديدًا. أما الإجراء HASH-SEARCH فلا نحتاج إلى تعديله، لأنه سيتخطى قيم DELETED أثناء البحث. ولكن، حين نستخدم القيمة الخاصة DELETED، تصبح أزمنة البحث غير معتمدة على معامل التحميل α، ولهذا السبب يجري عمومًا اختيار السَّلْسَلَة بصفتها تقانة لتمييز التصادم حين يجب حذف مفاتيح.

نفترض في تحليلنا أن ***التلبيد منتظم uniform hashing***: وهذا يعني أن متتالية السبر لأي مفتاح لها الاحتمال نفسه في أن تأخذ أيًّا من $m!$ تبديلاً من $\langle 0, 1, \ldots, m-1 \rangle$. يُعَمِّمُ التلبيد المنتظم فكرة التلبيد المنتظم البسيط المعرَّف سابقًا إلى الحالة التي تنتج فيها دالة التلبيد ليس فقط عددًا مفردًا، بل متتالية سبر كاملة. إن التلبيد المنتظم الحقيقي صعب التحقيق، إلا أننا نستخدم عمليًّا تقريبات مناسبة (مثل التلبيد المضاعف، المُعَرَّف لاحقًا).

سنختبر ثلاث طرق شائعة الاستخدام لحساب متتاليات السبر المطلوبة في العنونة المفتوحة: السبر الخطي linear probing والسبر التربيعي quadratic probing والتلبيد المضاعف double hashing. تضمن هذه التقانات أن يكون $\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ تبديلاً من $\langle 0, 1, \ldots, m-1 \rangle$ لكل مفتاح k. ولكن، لا تلبي أيٌّ من هذه التقانات فرَضية التلبيد المنتظم، لأن أيًّا منها غير قادر على توليد أكثر من m^2 متتالية سبرٍ مختلفة (عوضًا عن $m!$ التي يتطلَّبها التلبيد المنتظم). ينجز التلبيد المضاعف أكبر عدد من متتاليات السبر، وكما قد يتوقع المرء، يبدو أنه يعطي أفضل النتائج.

السبر الخطي

لتكن لدينا دالة تلبيد عادية $h' : U \to \{0, 1, \ldots, m-1\}$، نسميها ***دالة تلبيد مساعدة auxiliary hash function***، تَستخدم طريقةُ ***السبر الخطي linear probing*** دالةَ التلبيد

$$h(k,i) = (h'(k) + i) \bmod m$$

حيث $i = 0, 1, \ldots, m-1$. ليكن لدينا المفتاح k، نسبر أولاً الشَّقْب $T[h'(k)]$، أي الشقب المعطى بوساطة دالة التلبيد المساعدة. ثم نسبر الشقب $T[h'(k)+1]$، وهكذا حتى الشقب $T[m-1]$. ثم نلتف عائدين إلى الشقوب $T[0], T[1], \ldots$ حتى نسبر أخيرًا الشقب $T[h'(k)-1]$. ولما كان الشقب الذي يبدأ عنده السبر هو الذي يحدِّد كامل متتالية السبر، فيوجد m متتالية سبر متمايزة فقط.

إن السبر الخطي سهل التنجيز، لكنه يعاني من مشكلة تُعْرَفُ ***بالعنقدة الأولية primary clustering***. إذ تنشأ تدريجيًّا سلاسلُ طويلة بسبب الشقوب المشغولة، تؤدي إلى زيادة زمن البحث الوسطي. وتَظهر العناقيد لأن الشقبَ الفارغ المسبوق بـ i شقبًا ممتلئًا سيُمْلأُ بعدها باحتمال $(i+1)/m$. تَنْزع السلاسل الطويلة بسبب الشقوب المشغولة لتصبح أطول، ويزداد بذلك زمن البحث الوسطي.

السبر التربيعي

يَستخدم ***السبر التربيعي quadratic probing*** دالة تلبيد من الشكل

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m \ , \tag{11.5}$$

حيث h' دالة تلبيد مساعدة، و c_1 و c_2 ثابتان مساعدان موجبان، و $i = 0, 1, \dots, m-1$. إن أول موقع سيجري سبره هو $T[h'(k)]$؛ والمواقع التالية التي سيتم سبرها هي إزاحات بقيم تعتمد بطريقة تربيعية على رقم السبر i. إن أداءَ هذه الطريقة أفضلُ كثيرًا من السبر الخطي، لكن حتى نستخدم كامل جدول التلبيد، تُقَيَّدُ قيم c_1 و c_2 و m. توضِّح المسألةُ 11-3 طريقةً لاختيار هذه الموسطات. وكذلك، إذا كان لِمفتاحَيْن موضعُ السبر البدائي نفسه، يكون لهما متتالية السبر نفسها، لأن $h(k_1, 0) = h(k_2, 0)$ ينطوي ضمنيًّا على $h(k_1, i) = h(k_2, i)$. تؤدي هذه الخاصية إلى شكل متزايد من العنقدة، يسمى ***العنقدة الثانوية secondary clustering***. وكما في السبر الخطي، يحدِّد السبرُ البدائي كامل المتتالية، وهكذا يُستَخْدَمُ m متتاليةً متمايزةً فقط.

التلبيد المضاعف

يتيح التلبيد المضاعف إحدى أفضل الطرائق المتاحة للعنونة المفتوحة لأن التباديل الناتجة تمتلك كثيرًا من خواص ***التباديل permutations*** المختارة عشوائيًا. يَستَخْدِمُ ***التلبيدُ المضاعف double hashing*** دالةَ تلبيد من الصيغة

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m \ ,$$

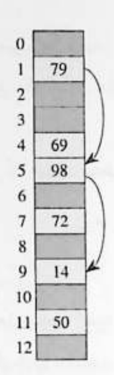
حيث h_1 و h_2 دالتا تلبيد مساعدتان. يبدأ السبر في الموقع $T[h_1(k)]$؛ وتكون مواقع السبر المتعاقبة هي إزاحة عن المواقع السابقة بمقدار القيمة $h_2(k)$، بالمقاس m. ومن ثم، وخلافًا لحالة السبر الخطي والتربيعي، يعتمد تتالي السبر هنا من جانبين على المفتاح k، لأنه قد يتغير موقع السبر الابتدائي، أو الإزاحة، أو كلاهما. يعطي الشكل 5.11 مثالاً على الإدراج باستخدام التلبيد المضاعف.

يجب أن تكون القيمة $h_2(k)$ وحجم جدول التلبيد بأكمله m أوليين فيما بينهما حتى يكون بالإمكان البحث في كامل جدول التلبيد. (انظر التمرين 11.4-4) وتكمن إحدى الطرق المناسبة لضمان هذا الشرط في جعل m قوة من قوى العدد 2 ولتصميم h_2 بحيث تُنتِج عددًا فرديًّا دومًا. ثمة طريقةٌ أخرى نجعل فيها m عددًا أوليًّا ونصمِّم h_2 بحيث تعيد دومًا عددًا صحيحًا موجبًا أقل من m. فمثلاً، يمكن أن نختار m عددًا أوليًّا ونجعل

$$h_1(k) = k \bmod m \ ,$$

$$h_2(k) = 1 + (k \bmod m') \ ,$$

حيث نختار m' أقل قليلاً من m (وليكن $m-1$). مثلاً، إذا كان $k = 123456$ و $m = 701$



الشكل 5.11 الإدراج باستخدام التلبيد المضاعف. لدينا هنا جدول تلبيد حجمه 13 و $h_1(k) = k \bmod 13$ و $h_2(k) = 1 + (k \bmod 11)$. ولما كان $14 \equiv 1 \pmod{13}$ و $14 \equiv 3 \pmod{11}$، فإننا نُدرج المفتاح 14 في الشَّقْب الفارغ 9، بعد أن نَفْحص الشقبَيْن 1 و 5 ونجد أنهما مشغولان.

و $m' = 700$، فلدينا $h_1(k) = 80$ و $h_2(k) = 257$، وبذلك نَسْبُرُ أولاً الموقع 80، ثم نفحص كل 257 شقب (بالمقاس m) إلى أن نجد المفتاح أو تُفْحَصَ كل الشقوب.

حينما يكون m عددًا أوليًّا أو من قوى العدد 2، يقدم التلبيدُ المضاعفُ تحسينات مقارنةً بالسبر الخطي والسبر التربيعي بأنه يَستَخدِمُ متتاليات عددها $\Theta(m^2)$، عوضًا عن $\Theta(m)$، لأن كل زوجٍ مُحْتَمَل $(h_1(k), h_2(k))$ يؤدي إلى متتالية سبر متمايزة. يَظهر إذن أن أداءَ التلبيد المضاعف، لمثل هذه القيم من m، قريبٌ جدًا من أداء السلوك "المثالي" للتلبيد المنتظم.

بالرغم من أنه يمكن من حيث المبدأ اختيار m مختلفة عن الأعداد الأولية وقوى العدد 2 لاستخدامها في التلبيد المضاعف، إلا أن ذلك يجعل إيجاد طريقة فعالة لتوليد $h_2(k)$ بحيث يكون أولياً مع m أكثر صعوبة. وأحد أسباب ذلك هو ضعف الكثافة النسبية لمثل هذه الأعداد الأولية مع m، المقدر بـ $\phi(m)/m$ (انظر المعادلة 24.31).

تحليل تلبيد العنونة المفتوحة

نُعَبِّر عن تحليلنا للعنونة المفتوحة، كما في تحليلنا للسَّلْسَلة، بدلالة معامل تحميل جدول التلبيد $\alpha = n/m$. طبعًا، في العنونة المفتوحة، يَشغل كلَّ شَقْبٍ عنصرٌ واحدٌ على الأكثر، ومن ثم فإن $n \le m$، وهذا يقتضي أن $\alpha \le 1$.

نفترض أننا نستخدم التلبيد المنتظم. وتبعاً لهذه الفرضية المثالية، تكون متتالية السبر $\langle h(k,0), h(k,1), \dots, h(k,m-1) \rangle$ المستخدمة لإدراج أي مفتاح k أو البحث عنه متساوية الاحتمال في

أن تكون أيًّا من التباديل $(0,1,\ldots,m-1)$. بالطبع، لكل مفتاح معطى متتالية سبر ثابتة وحيدة مرتبطة به؛ ما نقصده هنا، هو أنه بالأخذ بالحسبان توزع الاحتمال على فضاء المفاتيح وتطبيق دالة التلبيد على المفاتيح، تكون أية متتالية سبر ممكنة متساوية الاحتمال.

نحلل الآن عدد مرات السبر المتوقعة لتلبيد باستخدام العنونة المفتوحة بفرض أن التلبيد منتظم، مبتدئين بتحليل عدد حالات السبر المنفذة في حالة بحث غير ناجح.

مبرهنة *6.11*

ليكن لدينا جدول تلبيد ذو عنونة مفتوحة مُعامِلُ تحميله $\alpha = n/m < 1$، عندها يساوي العدد المتوقع من السُّبُور [جمع سَبْر] في بحثٍ غير ناجح $1/(1-\alpha)$ على الأكثر، وذلك بافتراض أن التلبيد منتظم.

البرهان إنَّ كلَّ سبرٍ – في بحث غير ناجح – ما عدا السبر الأخير يَنْفُذُ إلى شَقْبٍ مشغولٍ لا يحتوي المفتاح المطلوب، والشقب الأخير المختبَر فارغ. نعرّف المتحول العشوائي X على أنه عدد السبور المنفَّذة في بحثٍ غير ناجح، ونعرّف الحدث A_i، حيث $i = 1,2,\ldots$ على أنه الحدث المتمثل في حدوث السبر ذي الرقم i، والذي يسبر شقباً مشغولاً. فيكون الحدث $\{X \geq i\}$ هو تقاطع الأحداث $A_1 \cap A_2 \cap \cdots \cap A_{i-1}$. سَنَحُدُّ $\Pr\{X \geq i\}$ بحدّ المقدار $\Pr\{A_1 \cap A_2 \cap \cdots \cap A_{i-1}\}$. وباستخدام التمرين ت.2-5،

$$\Pr\{A_1 \cap A_2 \cap \cdots \cap A_{i-1}\} = \Pr\{A_1\} \cdot \Pr\{A_2 | A_1\} \cdot \Pr\{A_3 | A_1 \cap A_2\} \cdots \Pr\{A_{i-1} | A_1 \cap A_2 \cap \cdots \cap A_{i-2}\} .$$

ولما كان لدينا n عنصرًا و m شَقْبًا، فإن $\Pr\{A_1\} = n/m$. في حالة $j > 1$، فإن احتمال وجود سبر ذي ترتيب j لشقب مشغول هو $(n-j+1)/(m-j+1)$، وذلك بافتراض أنَّ أول $j-1$ سبرًا كانت لشُقُوبٍ مشغولة. ينتج هذا الاحتمال لأننا قد نجد عنصرًا من الـ $(n-(j-1))$ عنصرًا المتبقية، في شَقْبٍ من الـ $(m-(j-1))$ شقبًا غير المفحوصة، وبافتراض أن التلبيد منتظم، يكون الاحتمال هو نسبة هاتين الكميتين. وبملاحظة أن $n < m$ يقتضي $(n-j)/(m-j) \leq n/m$ لكل j بحيث $0 \leq j < m$، لدينا لكل i حيث $1 \leq i \leq m$،

$$\begin{aligned}
\Pr\{X \geq i\} &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \\
&\leq \left(\frac{n}{m}\right)^{i-1} \\
&= (\alpha)^{i-1} .
\end{aligned}$$

الآن نستخدم المعادلة (ت.25) لحدِّ العدد المتوقع من السبور:

$$E[X] = \sum_{i=1}^{\infty} \Pr\{X \geq i\}$$

$$\leq \sum_{i=1}^{\infty} \alpha^{i-1}$$
$$= \sum_{i=0}^{\infty} \alpha^{i}$$
$$= \frac{1}{1-\alpha} \ . \qquad \blacksquare$$

لهذا الحد من $\cdots + 1/(1-\alpha) = 1 + \alpha + \alpha^2 + \alpha^3$ تفسير بديهي: نُنَفِّذُ دائمًا أول سبر. وباحتمال تقريبي α، يَجِد السبرُ الأولُ شَقْبًا مشغولاً بحيث نحتاج إلى إجراء سبرٍ ثانٍ. وباحتمال تقريبي α^2، يكون الشقبان الأوّلان مشغولَيْن بحيث نجري سبرًا ثالثًا، وهكذا.

إذا كان α ثابتًا، فإن المبرهنة 6.11 تتنبأ بأن بحثًا غير ناجح ينفَّذ في زمن $O(1)$. فمثلاً إذا كان جدول التلبيد نصف ممتلئ، فالعدد المتوسط للسبور في بحثٍ غير ناجح هو على الأكثر $1/(1-0.5) = 2$. أما إذا كان ممتلئًا بنسبة 90 بالمئة، فيكون العدد المتوسط للسبور على الأكثر $1/(1-0.9) = 10$.

تعطينا المبرهنة 6.11 أداء الإجراء HASH-INSERT مباشرةً تقريبًا.

نتيجة 7.11

يتطلب إدراج عنصر في جدول تلبيد عنونة مفتوحة معامل تحميله α، $1/(1-\alpha)$ سبرًا وسطيًّا على الأكثر، وذلك بافتراض أن التلبيد منتظم.

البرهان نُدْرِجُ عنصرًا في حال تَوَفُّرَ فراغ في الجدول فقط، وبذلك يكون $\alpha < 1$. ويتطلَّب إدراجُ مفتاحٍ بحثًا غيرَ ناجح يتبعه وضْع المفتاح في أول شقب نجده فارغًا. ومن ثَمَّ، يكون العدد المتوقع للسبور $1/(1-\alpha)$ على الأكثر. ■

علينا العمل أكثر قليلاً لحساب عدد السبور المتوقع في حالة بحث ناجح.

مبرهنة 8.11

إذا كان لدينا جدول تلبيد عنونة مفتوحة معاملُ تحميله $\alpha < 1$، فإن عدد السبور المتوقع في البحث الناجح هو على الأكثر

$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha} \ ,$$

وذلك بافتراض أن التلبيد منتظم، وأن احتمالات البحث عن أي مفتاح في الجدول متساوية.

البرهان يُعيدُ البحثُ عن مفتاحٍ ما k إنتاجَ متتالية السبر نفسها، التي اتُّبِعَت أثناء إدراج العنصر ذي المفتاح k. وحسب النتيجة 7.11، إذا كان k هو المفتاح ذو الترتيب $(i+1)$ الذي أُدرجَ في الجدول، فعدد السبور

المتوقع للبحث عن المفتاح k هو على الأكثر $1/(1-i/m) = m/(m-i)$. إن حساب متوسط كل المفاتيح n في جدول التلبيد يعطينا العدد المتوقع للسبور في حالة بحث ناجح:

$$\begin{aligned}
\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} &= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} \\
&= \frac{1}{\alpha}\sum_{k=m-n+1}^{m}\frac{1}{k} \\
&\leq \frac{1}{\alpha}\int_{m-n}^{m}(1/x)dx \qquad \text{((باستخدام المتراجحة (أ.12))} \\
&= \frac{1}{\alpha}\ln\frac{m}{m-n} \\
&= \frac{1}{\alpha}\ln\frac{1}{1-\alpha}\ .
\end{aligned}$$

■

إذا كان جدول التلبيد نصف ممتلىء، يكون عدد السبور المتوقع في بحث ناجح أقل من 1.387. وإذا كان جدول التلبيد ممتلئًا بنسبة 90 بالمئة، يكون عدد السبور المتوقع أقل من 2.559.

تمارين

1-4.11

ادرس إدراج المفاتيح 59 ,88 ,17 ,28 ,15 ,4 ,31 ,22 ,10 في جدول تلبيد طوله $m = 11$ باستخدام العنونة المفتوحة، حيث دالة التلبيد المساعدة $h'(k) = k$. وضّح نتيجة إدراج هذه المفاتيح باستخدام السبر الخطي، وباستخدام السبر التربيعي حيث $c_1 = 1$، و $c_3 = 3$، وباستخدام التلبيد المضاعف حيث $h_1(k) = k$ و $h_2(k) = 1 + (k \bmod (m-1))$.

2-4.11

اكتب شبه رماز للإجراء HASH-DELETE كالملخّص في النص، وعَدِّل الإجراء HASH-INSERT كي يعالج القيمة الخاصة DELETED.

3-4.11

ادرس جدول تلبيد بعنونة مفتوحة حيث التلبيد منتظم. أعطِ حدودًا عليا لعدد السبور المتوقع في بحث غير ناجح، ولعدد السبور المتوقع في بحث ناجح حين يكون معامل التحميل 3/4، ثم حين يكون 7/8.

4-4.11 ★

افترض أننا نستخدم التلبيد المضاعف لتمييز التصادمات– أي نستخدم دالة التلبيد $h(k,i) = (h_1(k) + ih_2(k)) \bmod m$. بيِّن أنه إذا كان $d \geq 1$ هو القاسم المشترك الأعظم لكل من m و $h_2(k)$ في حالة مفتاح ما k، فإن بحثًا غير ناجح عن المفتاح k يختبر حتى الشَّقْب ذي الترتيب $1/d$ من

جدول التلبيد قبل أن يعود إلى الشقب $h_1(k)$. وأنه إذا كان $d = 1$ و m و $h_2(k)$ أوَّليَّين فيما بينهما، فإن البحث قد يَفحص كاملَ جدول التلبيد. (*تلميح:* انظر الفصل 31.)

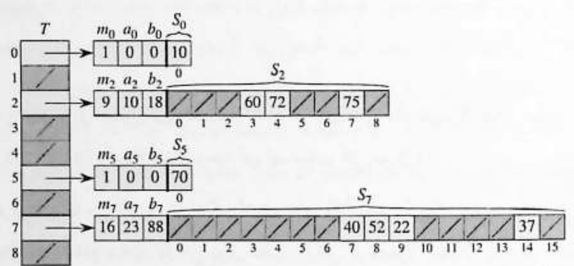
★ *5-4.11*

ادرس جدول تلبيد بعنونة مفتوحة معاملُ تحميله α. أوجد القيمة α المغايرة للصفر التي يكون عندها العدد المتوقع للسبور لبحث غير ناجح مساوياً لضعف العدد المتوقع للسبور في بحث ناجح. استخدم الحدود العليا المعطاة في المبرهنتين 6.11 و 8.11 لهذه الأعداد المتوقعة للسبور.

★ 5.11 التلبيد الكامل

إضافةً إلى أن التلبيدَ هو على الأغلب خيارٌ جيد من أجل أدائه الرائع في المتوسط، فيمكن أن يوفر أيضًا أداءً رائعًا في *أسوأ الحالات* عندما تكون مجموعة المفاتيح ***ساكنة static***: وذلك لأنه بمجرد تخزين المفاتيح في الجدول، فإن مجموعة المفاتيح لا تتغير أبدًا. وثمة تطبيقاتٌ لها، بصورة طبيعية، مجموعاتُ مفاتيح ساكنة: كمجموعة الكلمات المحجوزة في لغة برمجة، أو الأسماء على قرص متراص CD-ROM. نسمي تقانة التلبيد ***تلبيدًا كاملاً perfect hashing*** إذا كان عدد مرات النفاذ إلى الذاكرة المطلوبة لتنجيز بحث في أسوأ الحالات هو $O(1)$.

لإيجاد أسلوبِ تلبيدٍ كامل، نستخدم تلبيدًا ذا مستويين، كلٌّ منهما تلبيدٌ شامل. يوضح الشكل 6.11 هذا النهج.



الشكل 6.11 استخدام التلبيد الكامل لتخزين المجموعة $K = \{10, 22, 37, 40, 52, 60, 70, 72, 75\}$. دالة التلبيد الخارجية هي $h(k) = ((ak + b) \bmod p) \bmod m$، حيث $a = 3$ و $b = 42$ و $p = 101$ و $m = 9$. مثلاً، $h(75) = 2$، وهكذا يتلبد المفتاح 75 في الشقب 2 للجدول T. يخزِّن جدولُ تلبيدٍ ثانوي S_j جميعَ المفاتيح التي تتلبد في الشقب j. حجم الجدول S_j هو $m_j = n_j^2$، ودالة التلبيد المرافقة هي $h_j(k) = ((a_j k + b_j) \bmod p) \bmod m_j$. ولما كان $h_2(75) = 7$، فإن المفتاح 75 يخزَّن في الشقب 7 لجدول التلبيد الثانوي S_2. لا تحدث أية تصادمات في أيٍّ من جداول التلبيد الثانوية. وبذلك، يستغرق البحث زمنًا ثابتًا في أسوأ الحالات.

أما المستوى الأول فهو نفسه التلبيدُ مع السَّلْسَلة جوهريًّا: نُلَبِّد الـ n مفتاحًا في m شَقْبًا باستخدام دالة تلبيد h نختارها بعناية من جماعة دوال تلبيد شاملة.

ولكن، بدلاً من تكوين لائحة من المفاتيح تتلبد في الشقب j، نستخدم ***جدول تلبيد ثانوي*** ***secondary hash table*** S_j دالةُ تلبيدِهِ المرافقة هي h_j. وباختيار دوال التلبيد h_j بعناية، نضمن عدم وجود تصادمات في المستوى الثاني.

ولكي نضمن عدم وجود تصادمات في المستوى الثاني، نحتاج أن نجعل m_j، وهو حجم جدول التلبيد S_j، يساوي مربع العدد n_j الذي هو عدد المفاتيح التي تتلبد في الشقب j. وقد تظن أن الاعتماد التربيعي لـ m_j على n_j قد يبدو أنه يؤدي على الأرجح إلى أن يصبح متطلب التخزين الكلي مفرطاً في الكبر، غير أننا سنبين أنه باختيار دالة تلبيد المستوى الأول بعناية، يمكننا الحد من المقدار الكلي المتوقع لفضاء الذاكرة المستخدم إلى رتبة $O(n)$.

نستخدم دوال تلبيد اختيرت من صفوفٍ شاملة من دوال التلبيد المذكورة في المقطع 11.3-3. تأتي دالة تلبيد المستوى الأول من الصف $\mathcal{H}_{pm}$، حيث – كما في المقطع 11.3-3 – p عددٌ أولي أكبر من قيمة أيِّ مفتاح. يعاد تلبيد هذه المفاتيح الملبَّدة أصلاً في الشقب j في جدول تلبيد ثانوي S_jحجمه m_j باستخدام دالة التلبيد h_j التي نختارها من الصف $\mathcal{H}_{p,m_j}$.[3]

سنتابع العمل على مرحلتين. أولاً، نحدد كيفية التحقق من أن الجداول الثانوية لا تحتوي أي تصادمات. ثانيًا، نبيِّن أن الكمية المتوقعة من الذاكرة المستخدمة إجمالاً – لجدول التلبيد الأولي وجميع جداول التلبيد الثانوية – هي $O(n)$.

مبرهنة 9.11

افترض أننا خزَّنا n مفتاحًا في جدول تلبيد حجمه $m = n^2$ باستخدام دالة تلبيد h مختارة عشوائيًّا من صفٍّ شامل من دوال تلبيد، عندها يكون احتمال وجود أي تصادمات أقل من 1/2.

البرهان يوجد $\binom{n}{2}$ زوجًا من المفاتيح التي يمكن أن تتصادم؛ وكلُّ زوجٍ يمكن أن يتصادم باحتمال $1/m$ إذا اختيرت h عشوائيًّا من جماعةٍ شاملة $\mathcal{H}$ من دوال التلبيد. ليكن X متحولاً عشوائيًّا يَعُدُّ عدد التصادمات. عندما $m = n^2$، يكون العدد المتوقع للتصادمات

$$\mathrm{E}[X] = \binom{n}{2} \cdot \frac{1}{n^2}$$

---

[3] في حال $n_j = m_j = 1$، فإننا لا نحتاج حقيقة إلى دالة تلبيد للشقب j؛ فعندما نختار دالة تلبيد $h_{ab}(k) = ((ak + b) \bmod p) \bmod m_j$ لهذا الشقب، نجعل $a = b = 0$ فقط.

$$= \frac{n^2 - n}{2} \cdot \frac{1}{n^2}$$
$$< \frac{1}{2} .$$

(إن هذا التحليل شبيه بتحليل متناقضة عيد الميلاد في المقطع 4.5-1.) وبتطبيق متراجحة ماركوف (ت.30)، $\Pr\{X \geq t\} \leq \mathrm{E}[X]/t$، حيث $t = 1$، يكتمل البرهان. ■

في الحالة التي جرى وصفها في المبرهنة 9.11، حيث $m = n^2$، ينتج أن الاحتمال الأكبر لأن تكون دالة التلبيد h التي جرى اختيارها من $\mathcal{H}$ عشوائيًّا خالية من التصادمات. لتكن لدينا المجموعة K المؤلَّفة من n مفتاحًا (تَذَكَّر أن K ساكنة)، إن من السهل إيجاد دالة تلبيد h خالية من التصادمات بتجارب عشوائية قليلة.

ولكن، في حالة n كبيرة، يكون جدول التلبيد ذو الحجم $m = n^2$ مفرطاً في الكبر. لذا، نعتمد منهج التلبيد الثنائي المستوى، ونستخدم منهج المبرهنة 9.11 فقط لتلبيد العناصر داخل كل شَقْب. ونستخدم دالة تلبيد h خارجية، أو ذات مستوى أول، لتلبيد المفاتيح في $m = n$ شقبًا. عندها، إذا تَلَبَّد n_j مفتاحًا في شقب j، نستخدم جدول تلبيد ثانوي S_j حجمه $m_j = n_j^2$ لتوفير بحث ذي زمن ثابت خالٍ من التصادم.

نعود الآن إلى قضية التحقق من أن الذاكرة الإجمالية المستخدمة هي $O(n)$. لما كان m_j حجم جدول التلبيد الثانوي ذي الترتيب j ينمو تربيعيًّا مع عدد المفاتيح المخزنة n_j، فإننا نجازف في أن يكون حجم الذاكرة الإجمالية مفرطاً في الكبر.

إذا كان حجم جدول المستوى الأول $m = n$، فإن حجم الذاكرة المستخدمة في حالة جدول التلبيد الأولي يكون $O(n)$، وذلك لخزن الحجوم m_j لجداول التلبيد الثانوية، ولخزن الموسطات a_j و b_j التي تعرِّف دوالَ التلبيد الثانوية h_j المسحوبة من الصف $\mathcal{H}_{p,m_j}$ الوارد في المقطع 11.3-3 (ما عدا في حالة $n_j = 1$ نستخدم $a = b = 0$). تقدم المبرهنة والنتيجة التاليتان حدًّا لحجوم جميع جداول التلبيد الثانوية المتوقعة مجتمعة. ثمة نتيجة ثانية تَحُدُّ احتمال أن يكون حجم جميع جداول التلبيد الثانوية مجتمعة فوق خطي (يساوي أو يتجاوز $4n$ فعليًّا).

مبرهنة 10.11

افترض أننا نخزِّن n مفتاحًا في جدول تلبيد حجمه $m = n$ باستخدام دالة تلبيد h مختارة عشوائيًّا من صفٍّ شامل من دوال تلبيد. عندها، يكون لدينا

$$\mathrm{E}\left[\sum_{j=0}^{m-1} n_j^2\right] < 2n ,$$

حيث n_j عددُ المفاتيح المتلبدة في الشَّقْب j.

البرهان سنبدأ بالمتطابقة التالية، التي تتحقق في حالة أي عدد صحيح غير سالب a:

$$a^2 = a + 2\binom{a}{2} \ . \tag{6.11}$$

لدينا

$$\begin{aligned}
\mathrm{E}\left[\sum_{j=0}^{m-1} n_j^2\right] \\
&= \mathrm{E}\left[\sum_{j=0}^{m-1}\left(n_j + 2\binom{n_j}{2}\right)\right] && \text{(من المعادلة (6.11))} \\
&= \mathrm{E}\left[\sum_{j=0}^{m-1} n_j\right] + 2\mathrm{E}\left[\sum_{j=0}^{m-1}\binom{n_j}{2}\right] && \text{(حسب خطية التوقع)} \\
&= \mathrm{E}[n] + 2\mathrm{E}\left[\sum_{j=0}^{m-1}\binom{n_j}{2}\right] && \text{(من المعادلة (1.11))} \\
&= n + 2\mathrm{E}\left[\sum_{j=0}^{m-1}\binom{n_j}{2}\right] . && \text{(لأن } n \text{ ليست متحولاً عشوائيًا)}
\end{aligned}$$

لتقويم المجموع $\sum_{j=0}^{m-1}\binom{n_j}{2}$، نلاحظ أن العدد الكلي لأزواج المفاتيح في جدول التلبيد هو الذي يتصادم فقط. وبحسب خواص التلبيد الشامل، فإن القيمة المتوقعة لهذا المجموع هي على الأكثر

$$\begin{aligned}
\binom{n}{2}\frac{1}{m} &= \frac{n(n-1)}{2m} \\
&= \frac{n-1}{2} ,
\end{aligned}$$

لأن $m = n$. إذن،

$$\begin{aligned}
\mathrm{E}\left[\sum_{j=0}^{m-1} n_j^2\right] &\le n + 2\frac{n-1}{2} \\
&= 2n - 1 \\
&< 2n .
\end{aligned}$$

■

نتيجة 11.11

افترض أننا نخزِّن n مفتاحًا في جدول تلبيد حجمه $m = n$ باستخدام دالة تلبيد h مختارة عشوائيًّا من صفٍّ شامل من دوال التلبيد، وأننا نجعل حجمَ كلّ جدول تلبيد ثانوي $m_j = n_j^2$ حيث $j = 0, 1, \ldots, m-1$.

إذن يكون الحجم المتوقع من الذاكرة اللازمة لجميع جداول التلبيد الثانوية في منهج التلبيد الكامل أقل من $2n$.

البرهان لما كان $m_j = n_j^2$ حيث $j = 0, 1, \ldots, m-1$، فإن المبرهنة 10.11 تعطي

$$\mathrm{E}\left[\sum_{j=0}^{m-1} m_j\right] = \mathrm{E}\left[\sum_{j=0}^{m-1} n_j\right]$$

$$< 2n ,$$

وبذا يكتمل البرهان. ■

نتيجة 12.11

افترض أننا نخزّن n مفتاحًا في جدول تلبيد حجمه $m = n$ باستخدام دالة تلبيد h مختارة عشوائيًا من صفٍّ شامل من دوال التلبيد، وأننا نجعل حجمَ كلِّ جدول تلبيد ثانوي $m_j = n_j^2$ حيث $j = 0, 1, \ldots, m-1$. عندها يكون احتمال أن تساوي الذاكرة الكلية المستخدمة لجداول التلبيد الثانوية قيمة $4n$ أو تتجاوزها، أقلَّ من 1/2.

البرهان نطبق متراجحة ماركوف (ت.30) ثانيةً، $\Pr\{X \geq t\} \leq \mathrm{E}[X]/t$، ولكن هذه المرة على المتراجحة (7.11)، حيث $X = \sum_{j=0}^{m-1} m_j$ و $t = 4n$:

$$\Pr\left\{\sum_{j=0}^{m-1} m_j \geq 4n\right\} \leq \frac{\mathrm{E}\left[\sum_{j=0}^{m-1} m_j\right]}{4n}$$

$$< \frac{2n}{4n}$$

$$= \frac{1}{2} .$$ ■

نلاحظ من النتيجة 12.11، أننا إذا فحصنا القليل من دوال التلبيد المختارة عشوائيًا من جماعةٍ شاملة، فسنجد سريعًا دالة تلبيد تَستخدم حجمًا معقولاً من الذاكرة.

تمارين

★ ***1-5.11***

افترض أننا ندرج n مفتاحًا في جدول تلبيد حجمه m باستخدام العنونة المفتوحة وتلبيد منتظم. وليكن $p(n, m)$ احتمال ألا يحدث أي تصادم. بيِّن أن $p(n, m) \leq e^{-n(n-1)/2m}$. (*تلميح:* انظر المعادلة (12.3).) ناقش أنه عندما تتجاوز n القيمة $\sqrt{m}$، يتناهى احتمال تجنب التصادمات إلى الصفر بسرعة.

مسائل

1-11 حد أطول سبر للتلبيد

افترض أننا نَسْتَخْدِمُ جدول تلبيد ذا عنونة مفتوحة حجمه m لتخزين $n \le m/2$ بندًا.

أ. بفرض أن التلبيد منتظم، بَيِّنْ، في حالة $i = 1, 2, \ldots, n$، أن احتمال أن تحتاج عمليةُ الإدراج ذات الترتيب i إلى أكثر من k سبرًا بالضبط هو 2^{-k} على الأكثر.

ب. بيِّن في حالة $i = 1, 2, \ldots, n$، أنَّ احتمال أن تحتاج عمليةُ الإدراج ذات الترتيب i إلى أكثر من $2 \lg n$ سبرًا هو $O(1/n^2)$.

لنُشِر بـ X_i إلى المتحول العشوائي الذي يرمز إلى عدد السُّبُور [جمع سَبْر] اللازمة لعملية الإدراج ذات الترتيب i. وقد وجدت في الجزء (ب) أن $\Pr\{X_i > 2 \lg n\} = O(1/n^2)$. افترض أن المتحول العشوائي $X = \max_{1 \le i \le n} X_i$ يرمز للعدد الأعظمي للسبور اللازمة لـ n عملية إدراج.

ت. بيِّن أن $\Pr\{X > 2 \lg n\} = O(1/n)$.

ث. بيّن أن الطول المتوقع $\mathrm{E}[X]$ لأطول متتاليةِ سبرٍ هو $O(\lg n)$.

2-11 حدُّ حجمِ شَقْبٍ عند استخدام السَّلسَلَة

افترض أن لدينا جدول تلبيد يحتوي n شَقْبًا، وأنه جرى تمييز التصادمات بالسَّلْسَلة، وافترض أنه جرى إدراج n مفتاحًا في الجدول. لكل مفتاحٍ احتمالٌ متساوٍ في أن يلبَّد في أيِّ شَقْب. وليكن M عدد المفاتيح الأعظم في أيِّ شَقْبٍ بعد أن تُدرَج كل المفاتيح. أثبت أن القيمة العظمى لـ $\mathrm{E}[M]$، توقُّع M، المتوقعة هي $O(\lg n / \lg \lg n)$.

أ. ناقش أن يكون الاحتمال Q_k، ليتلبد k مفتاحًا تمامًا في شَقْبٍ محدَّدٍ يعطى بالعلاقة

$$Q_k = \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k} .$$

ب. ليكن P_k احتمال أن تكون $M = k$، أي، احتمال أن يكون الشقب الذي يحتوي غالبية المفاتيح يحتوي k مفتاحًا. بيّن أن $P_k \le n Q_k$.

ت. استخدم تقريب Stirling، المعادلة (3.18)، لتبيان أن $Q_k < e^k / k^k$.

ث. بيِّن أنه يوجد ثابت $c > 1$ بحيث $Q_{k_0} < 1/n^3$ حيث $k_0 = c \lg n / \lg \lg n$. استنتج أن $P_k < 1/n^2$ لكلِّ $k \ge k_0 = c \lg n / \lg \lg n$.

ج. ناقش أن

$$\mathrm{E}[M] \leq \Pr\left\{M > \frac{c\lg n}{\lg\lg n}\right\} \cdot n + \Pr\left\{M \leq \frac{c\lg n}{\lg\lg n}\right\} \cdot \frac{c\lg n}{\lg\lg n} .$$

واستنتج أن $\mathrm{E}[M] = O(\lg n / \lg\lg n)$.

3-11 *السبر التربيعي*

افترض أنه قد طُلِب إلينا البحث عن مفتاح k في جدول تلبيد مواقعُهُ $0, 1, \ldots, m-1$، وافترض أن لدينا دالة تلبيد h تطابق فضاء المفتاح إلى المجموعة $\{0, 1, \ldots, m-1\}$. يكون نهج البحث كالتالي:

1. احسب القيمةَ $j = h(k)$، وضعْ $i = 0$.
2. اسبر الموقع i للمفتاح المرغوب k. أنْهِ البحثَ إذا وجدته، أو إذا كان هذا الموقع فارغًا.
3. ضَعْ $i = i + 1$. إذا كان i يساوي الآن m، يكون الجدول ممتلئًا، لذلك أنْهِ البحث، وإلا، ضَعْ $j = (i + j) \bmod m$، ثم عُد إلى الخطوة 2.

افترض أن m من قوى العدد 2.

أ. بيِّن أن هذا المخطط منتسَخ instance عن مخطط "السبر التربيعي" العام بإعطاء الثوابت المناسبة c_1 و c_2 للمعادلة (5.11).

ب. أثبت أن هذه الخوارزمية في أسوأ الحالات تفحص كلَّ موقعٍ في الجدول.

4-11 *التلبيد والاستيقان* Authentication

ليكن $\mathcal{H}$ صفًا من دوال التلبيد تُقابل فيه كلُّ دالة تلبيد $h \in \mathcal{H}$ المجموعةَ الشاملةَ من المفاتيح U إلى $\{0, 1, \ldots, m-1\}$. نقول إن $\mathcal{H}$ هو ***شامل من الطول*** ***k k −universal*** إذا تحقق أنه، لكلِّ متتاليةٍ ثابتة مكوَّنة من k مفتاحًا متمايزًا $\langle x^{(1)}, x^{(2)}, \ldots, x^{(k)} \rangle$ ولأيِّ h مختار عشوائيًّا من متتالية $\mathcal{H}$، فإن المتتالية $\langle h(x^{(1)}),\ h(x^{(2)}), \ldots, h(x^{(k)}) \rangle$ متساوية الاحتمال في أن تكون أيًّا من m^k متتاليةً من الطول k حيث تُسحَب العناصر من $\{0, 1, \ldots, m-1\}$.

أ. بيِّن أنه إذا كانت جماعةُ دوالِّ التلبيد $\mathcal{H}$ شاملةً من الطول 2 universal 2−، فإن $\mathcal{H}$ شاملة.

ب. افترض أن المجموعة الشاملة U هي مجموعةٌ مكوَّنة من قيمٍ ذات n مركبة، مسحوبة من $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$، حيث p أولي. خذ عنصرًا $x = \langle x_0, x_1, \ldots, x_{n-1} \rangle \in U$. عرِّف لأي n مركبة $a = \langle a_0, a_1, \ldots, a_{n-1} \rangle \in U$ دالةَ التلبيد h_a حيث:

$$h_a(x) = \left(\sum_{j=0}^{n-1} a_j x_j\right) \bmod p \ .$$

ليكن الصف $\mathcal{H} = \{h_a\}$. بيِّن أنَّ $\mathcal{H}$ شاملٌ، ولكن ليس شاملاً من الطول 2 universal 2 ‒ 2.
(تلميح: أوجد مفتاحًا تُنْتِجُ له جميع دوال التلبيد في $\mathcal{H}$ القيمة نفسها.)

ت. افترض أننا عدّلنا $\mathcal{H}$ قليلاً من الجزء (ب): بحيث يكون لأيِّ $a \in U$ ولأيِّ عنصر $b \in \mathbb{Z}_p$ نعرّف

$$h'_{ab}(x) = \left(\sum_{j=0}^{n-1} a_j x_j + b\right) \bmod p$$

و $\mathcal{H}' = \{h'_{ab}\}$ ناقش كون $\mathcal{H}'$ شاملاً من الطول 2. (تلميح: افترض عنصرين ثابتين $x \in U$ و $y \in U$ بحيث $x_i \neq y_i$ لقيمةٍ ما i. ماذا يحصل لكل من $h'_{ab}(x)$ و $h'_{ab}(y)$ حين تتغير قيم a_i و b ضمن $\mathbb{Z}_p$؟)

ث. افترض أن Bob و Alice اتفقا سرًّا على دالة تلبيد h من جماعةٍ من دوال التلبيد $\mathcal{H}$ شاملة ذات الطول 2. كلُّ دالةٍ $h \in \mathcal{H}$ تقابل بين فضاء المفاتيح U و $\mathbb{Z}_p$، حيث p أوليّ. فيما بعد، ترسل Alice رسالة m إلى Bob بوساطة الإنترنت، حيث $m \in U$. وستُتيَقَّن هذه الرسالة لـ Bob أيضًا بإرسال لصيقة استيقان $t = h(m)$، ويفحص Bob كون الزوج (m, t) الذي استقبله يحقق $t = h(m)$. افترض أن خصمًا سيعترض مسار (m, t) ويحاول أن يخدع Bob بالاستعاضة عن هذا الزوج بزوج آخر (m', t'). ناقش أن يكون احتمال أن ينجح الخصم في خداع Bob بقبول الزوج (m', t') هو على الأكثر $1/p$، بقطع النظر عن قدرة الحساب التي يمتلكها الخصم، وحتى لو كان الخصم يعرف عائلة دوال التلبيد المستخدمة $\mathcal{H}$.

ملاحظات الفصل

Knuth [211] و Gonnet [145] مرجعان ممتازان عن تحليل خوارزميات التلبيد. يُرجِعُ Knuth اختراعَ جداول التلبيد وطريقةَ السَّلْسَلة في تمييز التصادمات إلى H. P. Luhn (1953). وفي الوقت نفسه تقريبًا أوجد G. M. Amdahl فكرة العنونة المفتوحة.

قدَّم Carter و Wegman مفهومَ الصفوف الشاملة لدوال التلبيد في عام 1979 [59].

طوَّر كلٌّ من Fredman و Komlós و Szemerédi [112] منهجَ التلبيد الكامل للمجموعات الساكنة الذي عرض في المقطع 5.11. ووسَّعَ Dietzfelbinger وآخرون [87] طريقتهم للمجموعات الديناميكية، ومعالجة عمليات الإدراج والحذف في زمنٍ مُتَوَقَّعٍ مُخَمَّد $O(1)$.

12 أشجار البحث الثنائية

تدعم بنيةُ معطيات أشجار البحث الكثيرَ من العمليات على المجموعة الديناميكية، ويشمل ذلك عمليةَ البحث SEARCH، وإيجاد القيمة الصغرى MINIMUM، وإيجاد القيمة العظمى MAXIMUM، وإيجاد السَّابق PREDECESSOR، وإيجاد اللاحق SUCCESSOR، وعملية الإدراج INSERT والحذف DELETE. وبذلك، يمكننا استخدام شجرة البحث كمعجم وكرتل ذو أولوية priority queue على حدٍّ سواء.

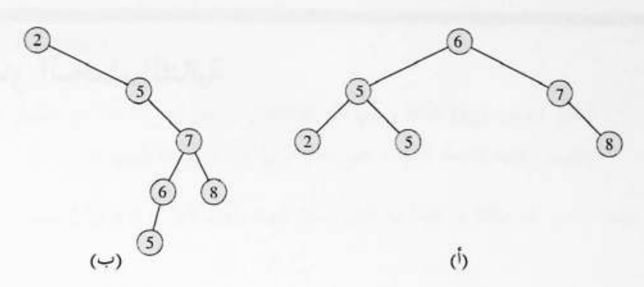
تستغرق العملياتُ الأساسيةُ على شجرة بحثٍ ثنائية زمنًا يتناسب مع ارتفاع الشجرة. وفي حالة شجرةٍ ثنائية كاملة من n عقدة، يستغرق زمنُ تنفيذ مثل هذه العمليات $\Theta(\lg n)$ في أسوأ الحالات. ولكن إذا كانت الشجرةُ سلسلةً خطِّية من n عقدة، فإن العمليات نفسها تستغرق زمنًا $\Theta(n)$، في أسوأ الحالات. وسنرى في المقطع 4.12 أن الارتفاع المتوقع لشجرة بحث ثنائية مبنيَّةٍ عشوائيًا هو $O(\lg n)$، وبذلك تستغرق العملياتُ على المجموعة الديناميكية الأساسية على شجرة كهذه زمنًا وسطيًا $\Theta(\lg n)$.

ومن الناحية العملية، لا يمكننا دومًا أن نضمن أن تكون أشجارُ البحث الثنائية قد بُنيت عشوائيًا، إلا أنه يمكننا إيجاد متغيرات (نسخ مُعدلة) من أشجار البحث الثنائية نضمن أن يكون أداؤها على العمليات الأساسية، في أسوأ الحالات، جيدًا. يعرض الفصل 13 مثالاً عن مثل هذه المتغيرات، وهو الأشجار الحمراء-السوداء red-black trees، التي ارتفاعها $O(\lg n)$. ويعرض الفصل 18 الأشجار المعممة B-trees، التي تُعتَبَر جيدة بصفةٍ خاصة للحفاظ على قواعد المعطيات على قرص خزن ثانوي.

بعد عرض الخصائص الأساسية لأشجار البحث الثنائية، تبيِّن المقاطعُ التالية كيف يمكن أن نجوب شجرةَ بحث ثنائية لطباعة قيمها بترتيب مفروز، وكيف نبحث عن قيمة في شجرة بحث ثنائية، وكيف نوجد العنصر الأصغر أو الأكبر، وكيف نوجد لاحق عنصرٍ ما وسابقه، وكيف ندرج في شجرة بحث ثنائية أو نحذف منها. يُظهر الملحق -ب- الخصائص الرياضية الأساسية للأشجار.

1.12 ما هي شجرة البحث الثنائية؟

تَنتظم شجرةُ البحث الثنائية، كما يوحي اسمها، على شكل شجرةٍ ثنائية، كما هو مبين في الشكل 1.12. يمكننا تمثيل هذه الشجرة ببنية معطيات مترابطة تؤلِّف كلُّ عقدةٍ فيها غرضًا. وتحتوي كلُّ عقدة، إضافة إلى



الشكل 1.12 أشجار بحث ثنائية. إن المفاتيح في الشجرة الفرعية اليسرى لأي عقدة x، تساوي على الأكثر $x.key$؛ والمفاتيح في الشجرة الفرعية اليمنى من x، تساوي $x.key$ على الأقل. يمكن أن تمثّل أشجارٌ بحث ثنائية مختلفة مجموعةَ القيم نفسَها. يتناسب زمن التنفيذ، في أسوأ الحالات، لأغلب عمليات البحث في الشجرة، مع ارتفاع الشجرة. (أ) شجرة بحث ثنائية من 6 عقد ارتفاعها 2. (ب) شجرة بحث ثنائية أقل فعالية ارتفاعها 4 وتحتوي المفاتيح نفسها.

المفتاح key والمعطيات التابعة satellite data، على الواصفات يسار $left$، ويمين $right$ و p التي تشير إلى العقد المقابلة للابن الأيسر وللابن الأيمن وللأب، على الترتيب. عندما يغيب الابن أو الأب، يحتوي الواصف الموافق القيمة NIL. إن عقدة الجذر root node هي العقدة الوحيدة في الشجرة التي تمتلك حقل أبٍ يساوي NIL.

تُخزن المفاتيح في شجرة بحث ثنائية دومًا بحيث تحقق ***خاصيةَ شجرة البحث الثنائية binary-search-tree property***:

لتكن x عقدة في شجرة بحث ثنائية. فإذا كانت y عقدةً في الشجرة الفرعية اليسرى لـ x، فإن $y.key \leq x.key$. وإذا كانت y عقدة في الشجرة الفرعية اليمنى لـ x، فإن $y.key \geq x.key$.

وهكذا فإن مفتاح الجذر، في الشكل 1.12(أ)، يساوي 6، والمفاتيح 2 و 5 و 5 في شجرتها الفرعية اليسرى ليست أكبر من 6. والمفاتيح 7 و 8 في الشجرة الفرعية اليمنى ليست أصغر من 6. وهذه الخاصية نفسها محققة لكل عقدة في الشجرة. على سبيل المثال، إن المفتاح 5 في الابن الأيسر للجذر ليس أصغر من المفتاح 2 في الشجرة الفرعية اليسرى لتلك العقدة وليس أكبر من المفتاح 5 في الشجرة الفرعية اليمنى.

تسمح لنا خاصية شجرة البحث الثنائية بطباعة كل المفاتيح في شجرة البحث الثنائية بترتيب مفروز باستخدام خوارزمية عَوْدِية بسيطة، تسمى ***تجوال في الشجرة وفق الترتيب البيني inorder tree walk***. سُميت هذه الخوارزمية هكذا لكونها تطبع مفتاح جذر شجرة فرعية بين طباعة القيم الموجودة في شجرتها

الفرعية اليسرى وطباعة تلك الموجودة في شجرتها الفرعية اليمنى. (بالطريقة نفسها، تَطبع خوارزميةُ ***تجوال في الشجرة وفق الترتيب السبقي preorder tree walk*** الجذرَ قبل القيم في أيٍّ من الشجرتين الفرعيتين، وتَطبع خوارزميةُ ***تجوال في الشجرة وفق الترتيب اللحقي postorder tree walk*** الجذرَ بعد القيم في أشجارها الفرعية.) ولاستعمال الإجراء التالي لطباعة جميع العناصر في شجرة بحثٍ ثنائية T، فإننا نستدعي INORDER-TREE-WALK($T.root$).

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2      INORDER-TREE-WALK(x.left)
3      print x.key
4      INORDER-TREE-WALK(x.right)
```

على سبيل المثال، تَطبع خوارزميةُ تجوال في شجرة وفق الترتيب الداخلي المفاتيحَ في كل من شجرتي البحث الثنائية من الشكل 1.12 بالترتيب 2، 5، 5، 6، 7، 8. وتُستنبط صحة الخوارزمية، مباشرة، من خاصية شجرة–البحث–الثنائية.

تستغرق الخوارزمية زمنًا $\Theta(n)$ لتجوب شجرة بحث ثنائية من n عقدة، لأن الإجراءَ، بعد الاستدعاء الأولي، يستدعي نفسَه على نحوٍ عَوْدِي مرتين بالضبط لكل عقدة في الشجرة: مرةً في حالة ابنها الأيسر ومرة في حالة ابنها الأيمن. وتُقدم المبرهنة التالية برهانًا منهجيًا على أن الخوارزمية تستغرق زمنًا خطيًا لإنجاز تجوال في شجرة وفق الترتيب البيني.

مبرهنة 1.12

إذا كان x جذرًا لشجرة فرعية من n عقدة، فإن استدعاء INORDER-TREE-WALK(x) يستغرق زمنًا $\Theta(n)$.

البرهان لِنُشر بـ $T(n)$ إلى الزمن الذي تستغرقه INORDER-TREE-WALK لدى استدعائها عند جذر شجرة فرعية من n عقدة. لمّا كان INORDER-TREE-WALK يزور جميع العقد n للشجرة الفرعية، كان لدينا $T(n) = \Omega(n)$. يبقى علينا إثبات أن $T(n) = O(n)$.

ولمّا كان INORDER-TREE-WALK يستغرق مدةً ثابتةً وصغيرة في شجرةٍ فرعيةٍ فارغة (عند اختبار $x \neq$ NIL)، فإن $T(0) = c$ حيث $c > 0$ ثابت ما.

لنفترض، في حالة $n > 0$، أنه جرى استدعاء الإجراء INORDER-TREE-WALK عند العقدة x التي لشجرتها الفرعية اليسرى k عقدة ولشجرتها الفرعية اليمنى $n - k - 1$ عقدة. إن الزمن اللازم لإنجاز INORDER-TREE-WALK(x) محدود بـ $T(n) \leq T(k) + T(n - k - 1) + d$، حيث d ثابت ما

موجب تمامًا ($d > 0$)، يعكس حدًا أعلى لزمن تنفيذ متن الإجراء INORDER-TREE-WALK(x) باستثناء الزمن المُستغرق في الاستدعاءات العودية.

نستخدم طريقة التعويض لنثبت أن $T(n) = O(n)$ وذلك ببرهان أن $T(n) \leq (c+d)n + c$. من أجل $n = 0$، لدينا $(c+d) \cdot 0 + c = c = T(0)$. ومن أجل $n > 0$، لدينا:

$$\begin{aligned} T(n) &\leq T(k) + T(n-k-1) + d \\ &= ((c+d)k + c) + ((c+d)(n-k-1) + c) + d \\ &= (c+d)n + c - (c+d) + c + d \\ &= (c+d)n + c\ , \end{aligned}$$

وهذا هو المطلوب. ■

تمارين

1-1.12

ارسم أشجار بحث ثنائية بارتفاع 2 و 3 و 4 و 5 و 6 لمجموعة المفاتيح $\{1,4,5,10,16,17,21\}$.

2-1.12

ما الفرق بين خاصية شجرة-البحث-الثنائية وخاصية الكومة وفق الأصغر min-heap (انظر الصفحة 154)؟ هل يمكن استخدام خاصية الكومة وفق الأصغر لطباعة مفاتيح شجرة من n عقدة بترتيب مفروز في زمن $O(n)$؟ اشرح الكيفية، أو اشرح لماذا لا يمكن ذلك.

3-1.12

أعطِ خوارزميةً غير عودية تنفِّذ تجوالًا في شجرة وفق الترتيب البيني. (*تلميح*: في حلٍّ سهلٍ يُستعمل مكدسٌ كبنية معطياتٍ رديفة. وفي حلٍّ أكثر تعقيدًا، لكنه أنيق، لا يُستعمل مكدس، وإنما يفترض أن بإمكاننا اختبار حالة تَساوي مؤشرين.)

4-1.12

أعطِ خوارزمياتٍ عَوْدِية تُنفذ تجوالاً في شجرة بالترتيب السبقي وبالترتيب اللحقي في زمن $\Theta(n)$ على شجرة من n عقدة.

5-1.12

ناقش مايلي: لمّا كان فرز n عنصرًا يستغرق في أسوأ الحالات زمنًا $\Omega(n \lg n)$ وفق نموذج المقارنة comparison model، فإن أي خوارزمية تعتمد على المقارنة لبناء شجرة بحث ثنائية من لائحة اعتباطية من n عنصرًا تستغرق في أسوأ الحالات زمنًا $\Omega(n \lg n)$.

2.12 استعلام شجرة بحث ثنائية

كثيرًا ما نحتاج إلى البحث عن مفتاح مخزن في شجرة بحث ثنائية. فإضافة إلى عملية البحث SEARCH، يمكن أن تدعم أشجارُ البحث الثنائية استعلاماتٍ مثل الأصغر MINIMUM، الأكبر MAXIMUM، اللاحق SUCCESSOR، السابق PREDECESSOR. وسندرس، في هذا المقطع، هذه العمليات ونبين كيف يمكن تنفيذ كل منها في زمن $O(h)$، على شجرة بحث ثنائية ما ارتفاعها h.

البحث

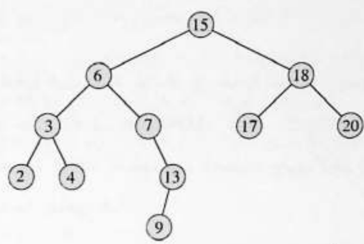
نستخدم الإجراء التالي للبحث عن العقدة التي تحتوي مفتاحًا محددًا في شجرة بحث ثنائية. فإذا كان لدينا مؤشر إلى جذر الشجرة ومفتاح k، فإن إجراء البحث في الشجرة TREE-SEARCH يعيد مؤشرًا إلى عقدةٍ مفتاحها k إن وُجدت، وإلا فإنه يعيد NIL.

```
TREE-SEARCH(x, k)
1  if x == NIL or k == x.key
2      return x
3  if k < x.key
4      return TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)
```

تبدأ الإجرائيةُ بحثَها من الجذر وتتبع مسارًا بسيطًا نازلاً باتجاه أسفل الشجرة، كما هو مبيَّن في الشكل 2.12. ثم تقارن، لكل عقدة x تصادفها، المفتاحَ k بالمفتاح $x.key$. فإذا تساوى المفتاحان انتهى البحث. وإذا كان k أصغر من $x.key$، استمر البحث في الشجرة الفرعية اليسرى لـ x، لأن خاصية شجرة-البحث-الثنائية تقتضي عدم إمكان تخزين k في الشجرة الفرعية اليمنى. وبالطريقة نفسها، إذا كان k أكبر من $x.key$، استمر البحث في الشجرة الفرعية اليمنى. تُشكل العقد التي تُصادَف خلال العودية مسارًا بسيطًا نازلاً من جذر الشجرة، ومن ثَمَّ فإن زمن تنفيذ TREE-SEARCH هو $O(h)$، حيث h ارتفاع الشجرة.

يمكننا كتابة الإجراء نفسه على نحو تكراري بـ "نشر unrolling" العودية إلى حلقة **while**. وهذه النسخة أكثر فعالية على معظم الحواسيب.

```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2      if k < x.key
3          x = x.left
4      else x = x.right
5  return x
```



الشكل 2.12 استعلامات على شجرة بحث ثنائية. للبحث عن المفتاح 13 في الشجرة فإننا نَتَّبع المسار 15 ← 6 ← 7 ← 13 بدءًا من الجذر. إن المفتاح الأصغر في الشجرة هو 2، ونجده بتعقُّب المؤشرات *اليسرى* left pointers من الجذر. والمفتاح الأكبر هو 20، ونجده بتعقب المؤشرات *اليمنى* right pointers من الجذر. إن لاحق العقدة ذات المفتاح 15 هي العقدة ذات المفتاح 17، لكونه المفتاح الأصغر في الشجرة الفرعية اليمنى لـ 15. لا تمتلك العقدة ذات المفتاح 13 شجرة فرعية يمنى، ومن ثَمَّ فإن لاحقها هو سلفها الأدنى الذي ابنه الأيسر هو أيضًا سلفٌ. وفي هذه الحالة، تكون العقدة ذات المفتاح 15 هي لاحقها.

الأصغر والأكبر

يمكننا دومًا العثور على عنصرٍ في شجرة بحث ثنائية مفتاحها أصغري بتتبُّع مؤشرات الأبناء *اليسرى* من الجذر إلى أن تُصادف القيمة NIL، كما هو مبين في الشكل 2.12. يعيد الإجراء التالي مؤشرًا إلى العنصر الأصغر في شجرة فرعية جذرها عند عقدة معطاة x، والذي نفترضه لا يساوي NIL:

```
TREE-MINIMUM(x)
1  while x.left ≠ NIL
2      x = x.left
3  return x
```

تَضمن خاصية شجرة–البحث–الثنائية صحةَ الإجراء TREE-MINIMUM. فإذا لم يكن للعقدة x شجرة فرعية يسرى، فإن المفتاح الأصغر في الشجرة الفرعية التي جذرها x هو $x.key$، لأن كل مفتاح في الشجرة الفرعية اليمنى لـ x هو على الأقل بِكِبَر $x.key$. أما إذا كان للعقدة x شجرة فرعية يسرى، فإن المفتاح الأصغر في الشجرة الفرعية التي جذرها x يقع في الشجرة الفرعية التي جذرها عند $x.left$، لأنه لا يوجد مفتاحٌ في الشجرة الفرعية اليمنى أصغر من $x.key$ وكلُّ مفتاح في الشجرة الفرعية اليسرى ليس أكبر من $x.key$.

إن شبه الرماز لـ TREE-MAXIMUM مشابه لـ TREE-MINIMUM(x).

```
TREE-MAXIMUM(x)
1  while x.right ≠ NIL
2      x = x.right
3  return x
```

يُنفَّذ كلا الإجراءين في زمن $O(h)$ لشجرة ارتفاعها h، لأن متتالية العُقَد العارضة، كما في إجراء TREE-SEARCH، تشكل مسارًا بسيطاً نازلاً من جذر الشجرة.

اللاحق والسابق

إذا كانت لدينا عقدة في شجرة بحث ثنائية، فقد نحتاج أحيانًا إلى إيجاد لاحقها في الترتيب المفروز المُحدد بواسطة تجوال في شجرة وفق الترتيب البيني. فإذا كانت جميع المفاتيح متمايزة، فإن لاحق العقدة x هي العقدة ذات أصغر مفتاح أكبر من $x.key$. تسمح لنا بنية شجرة البحث الثنائية بتحديد لاحق عقدة حتى بدون مقارنة المفاتيح. ويعيد الإجراء التالي لاحق العقدة x في شجرة بحث ثنائية إن كان موجودًا، ويعيد NIL إن كان مفتاح x هو الأكبر في الشجرة.

```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2      return TREE-MINIMUM(x.right)
3  y = x.p
4  while y ≠ NIL and x == y.right
5      x = y
6      y = y.p
7  return y
```

تجزّئ رماز الإجراء TREE-SUCCESSOR إلى حالتين. إذا لم تكن الشجرة الفرعية اليمنى للعقدة x فارغة، فإن لاحق x هي تمامًا أقصى عقدة يسرى leftmost في الشجرة الفرعية اليمنى، والتي نجدها في السطر 2 باستدعاء TREE-MINIMUM($x.right$). على سبيل المثال، إن لاحق العقدة ذات المفتاح 15 في الشكل 2.12 هي العقدة ذات المفتاح 17.

من ناحية أخرى، وكما يُطلب إليك في التمرين 2.12-6 بيانه، إذا كانت الشجرة الفرعية اليمنى للعقدة x فارغة وكان لـ x لاحق y، فإن y هو السلف الأدنى لـ x الذي ابنه الأيسر هو أيضًا سلف لـ x. إن لاحق العقدة ذات المفتاح 13، في الشكل 2.12، هي العقدة ذات المفتاح 15. ولإيجاد y، ما علينا إلا أن نصعد الشجرة ابتداءً من x حتى نلاقي عقدة تكون هي الابن الأيسر لأبيها؛ تعالج الأسطر 3-7 في الإجراء TREE-SUCCESSOR هذه الحالة.

إن زمن تنفيذ TREE-SUCCESSOR على شجرة ارتفاعها h هو $O(h)$، وذلك لأننا إما أن نتبع مسارًا بسيطًا باتجاه أعلى الشجرة، وإما أن نتبع مسارًا بسيطًا باتجاه أسفل الشجرة. إن الإجراء TREE-PREDECESSOR، المناظر لـ TREE-SUCCESSOR، يُنفذ أيضًا في زمن $O(h)$.

وحتى إن لم تكن المفاتيح متمايزة، فإننا نُعرِّف لاحق أي عقدة x وسابقها بالعقدة المُعادة من جراء الاستدعائين TREE-SUCCESSOR(x) و TREE-PREDECESSOR(x) على الترتيب.

بإيجاز، نكون قد برهنا المبرهنة التالية.

مبرهنة *2.12*

يمكننا تنجيز العمليات على مجموعة ديناميكية: البحث، والأصغر، والأكبر، واللاحق والسابق بحيث تُنفَّذ كل منها في زمن $O(h)$ على شجرة بحث ثنائية ارتفاعها h. ■

تمارين

1-2.12

بافتراض أن لدينا أعدادًا بين 1 و 1000 في شجرة بحث ثنائية، ونريد البحث عن العدد 363. أيٌّ من المتتاليات الآتية لا يمكن أن تكون متتالية من العقد المدروسة؟

أ. 2, 252, 401, 398, 330, 344, 397, 363.

ب. 924, 220, 911, 244, 898, 258, 362, 363.

ت. 925, 202, 911, 240, 912, 245, 363.

ث. 2, 399, 387, 219, 266, 382, 381, 278, 363.

ج. 935, 278, 347, 621, 299, 392, 358, 363.

2-2.12

اكتب نسخات عَوْدية لـ TREE-MINIMUM ولـ TREE-MAXIMUM.

3-2.12

اكتب إجراء البحث عن السابق TREE-PREDECESSOR.

4-2.12

يعتقد البرفسور بَنْيان Bunyan أنه قد اكتشف خاصية مدهشة لأشجار البحث الثنائية. افترضْ أن البحث عن مفتاح k في شجرة بحث ثنائية قد انتهى عند ورقة leaf. تأمل ثلاث مجموعات: A، المفاتيح الواقعة إلى يسار مسار البحث؛ و B، المفاتيح الواقعة على مسار البحث؛ و C، المفاتيح الواقعة إلى يمين مسار البحث. يَدعي البرفسور أن أي ثلاثة مفاتيح $a \in A$ و $b \in B$ و $c \in C$ يجب أن تحقق $a \leq b \leq c$. أعطِ أصغر مثال معاكس يمكن أن يدحض ادعاء البرفسور بَنْيان.

5-2.12

بيِّن أنه إذا كان لعقدةٍ في شجرة بحثٍ ثنائية ابنان، فليس للاحقها ابنٌ أيسر وليس لسابقها ابنٌ أيمن.

6-2.12

ادرس شجرة بحث ثنائية T مفاتيحها متمايزة. بيّن أنه إذا كانت الشجرة الفرعية اليمنى للعقدة x في الشجرة

T فارغة وكان y لاحق لـ x، فإن y هي السلف الأدنى lowest ancestor لـ x الذي ابنه الأيسر هو أيضًا سلف لـ x. (تذكَّر أن كلَّ عقدة هي سلف نفسها.)

7-2.12

ثمة طريقة أخرى لتنجيز تجوال بيني في شجرة بحث ثنائية مؤلَّفة من n عقدة، وذلك بإيجاد العنصر الأصغر في الشجرة باستدعاء TREE-MINIMUM، ومن ثم إجراء $n - 1$ استدعاء لـ TREE-SUCCESSOR. برهن أن هذه الخوارزمية تنفَّذ في زمن $\Theta(n)$.

8-2.12

برهن أنه أيًّا كانت العقدة التي نبدأ منها في شجرة بحث ثنائية ارتفاعها h، فإن k استدعاء متعاقبًا لـ TREE-SUCCESSOR يستغرق زمنًا $O(k + h)$.

9-2.12

لتكن T شجرة بحث ثنائية، مفاتيحها متمايزة، ولتكن x عقدة ورقة وليكن y أباها. بيّن أن $y.key$ هو إما المفتاح الأصغر في T الأكبر من $x.key$، وإما هو المفتاح الأكبر في T الأصغر من $x.key$.

3.12 الإدراج والحذف

تُسبِّب عمليتا الإدراج والحذف تغييرَ المجموعة الديناميكية المُمَثلة بشجرة بحث ثنائية. ولا بدَّ من تعديل بنية المعطيات كي يظهر هذا التغيير، ولكن بطريقة تسمح بالمحافظة على خاصية شجرة-البحث-الثنائية. وسنرى لاحقًا أن تعديل الشجرة لإدراج عنصر جديد هي عملية مباشرة نسبيًا، ولكن معالجة الحذف عملية أكثر تعقيدًا نوعًا ما.

الإدراج

نستخدم إجراء TREE-INSERT لإدراج قيمة جديدة v في شجرة بحث ثنائية T. يأخذ الإجراءُ العقدةَ z التي لها $z.key = v$ و $z.left =$ NIL و $z.right =$ NIL. يُعدِّل الإجراءُ الشجرةَ T وبعض واصفات z بحيث يُدرج z في مكان مناسب في الشجرة.

```
TREE-INSERT(T, z)
1  y = NIL
2  x = T.root
3  while x ≠ NIL
4      y = x
5      if z.key < x.key
6          x = x.left
```

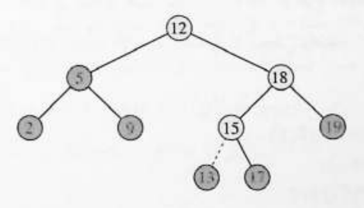
```
7       else x = x.right
8   z.p = y
9   if y == NIL
10      T.root = z   // Tree T was empty
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
```

يبين الشكل 3.12 آلية عمل الإجراء TREE-INSERT. يبدأ الإجراء TREE-INSERT، كما هو الحال في الإجرائيتين TREESEARCH و ITERATIVE-TREE-SEARCH، من جذر الشجرة ويرسم المؤشر x مسارًا بسيطًا نازلاً باحثًا عن NIL ليستعيض عنه بعنصر الدخل z. يحافظ الإجراء على ***مؤشر الأثر*** ***trailing pointer*** y كأبٍ لـ x. بعد الاستبداء، تسبِّب حلقةُ **while** في الأسطر 3-7 تحريكَ هذين المؤشرين باتجاه أسفل الشجرة، ويتجهان يسارًا أو يمينًا اعتمادًا على نتيجة مقارنة $z.key$ بـ $x.key$، حتى يصبح x مساويًا NIL. يشغل NIL هذا المكان الذي نرغب وضع العنصر z فيه. ونحتاج إلى مؤشر الأثر y، لأننا عندما نجد الـ NIL التي ينتمي إليها z، يكون البحث قد تقدم خطوة واحدة إلى ما بعد العقدة التي تحتاج إلى تغيير. تهيئ الأسطر 8-13 المؤشرات التي تسبِّب إدراجَ z.

وعلى نحو مماثل للعمليات الأولية الأخرى على أشجار البحث، يُنَفَّذ الإجراء TREE-INSERT في زمن $O(h)$ على شجرة ارتفاعها h.

الحذف

هناك ثلاث حالات أساسية للاستراتيجية الشاملة لحذف عقدة z من شجرة بحث ثنائية T، ولكن إحدى هذه الحالات تنطوي على شيءٍ من الدقة والتعقيد، كما سيتبيَّن لنا لاحقًا.



الشكل 3.12 إدراج عنصر مفتاحه 13 في شجرة بحث ثنائية. تشير العقد الخفيفة التظليل إلى المسار البسيط المنحدر من الجذر باتجاه الموضع الذي جرى فيه إدراج العنصر. يشير الخطُّ المتقطع إلى الوصلة في الشجرة التي أضيفت لإدراج العنصر.

- إذا لم يكن لـ z أبناء، فإننا ببساطة نزيلها بتغيير أبيها بالاستعاضة عن z بـ NIL كابن لها.
- إذا كان لـ z ابن وحيد، فإننا نرفع هذا الابن ليأخذ مكان z في الشجرة بتعديل أبي z للاستعاضة عن z بابن z.
- إذا كان لـ z ابنان، فإننا نوجد y لاحق z – الذي يجب أن يكون في الشجرة الفرعية اليمنى لـ z – ونجعل y تأخذ مكان z في الشجرة. ما تبقى من الشجرة الفرعية اليمنى الأصلية لـ z تصبح الشجرة الفرعية اليمنى الجديدة لـ y، وتصبح الشجرةُ الفرعيةُ اليسرى لـ z الشجرةَ الفرعيةَ اليسرى الجديدة لـ y. ويتمثل وجه التعقيد في هذه الحالة، كما سنرى، في الأهمية المترتبة على كون y الابن الأيمن لـ z.

يأخذ إجراءُ حذف عقدةٍ معطاةٍ z من شجرة بحث ثنائية T مؤشرَيْن إلى T وإلى z كمحدِّدَيْن arguments. يُرتِّب الإجراء حالاته بطريقة مختلفة قليلاً عن الحالات الثلاث المذكورة آنفًا، وذلك باعتبار الحالات الأربع المبينة في الشكل 4.12.

- إذا لم يكن لـ z ابن أيسر (الجزء (أ) من الشكل)، فإننا نستعيض عن z بابنه الأيمن، الذي من الممكن أن يكون NIL أو لا يكون. فإذا كان الابن الأيمن لـ z يساوي NIL، فإن هذه الحالة تُعَالَج مع الوضعية التي ليس فيها لـ z أبناء. أما إذا كان الابن الأيمن لـ z لا يساوي NIL، فإن هذه الحالة تُعالِج الوضعية التي فيها لـ z ابن وحيد، الذي هو ابنه الأيمن.
- إذا كان لـ z ابن وحيد، الذي هو ابنه الأيسر (الجزء (ب) من الشكل)، فإننا نستعيض عن z بابنه الأيسر.
- وفيما عدا ذلك، فإن لـ z ابن أيسر وابن أيمن. نوجد y لاحق z، الذي يقع في الشجرة الفرعية اليمنى لـ z وليس له ابن أيسر (انظر التمرين 12.2-5). نريد أن نَنْزِع y من مكانه الحالي ونضعه مكان z في الشجرة.
  - إذا كان y الابن الأيمن لـ z (الجزء (ت))، فإننا نستعيض عن z بـ y، تاركين الابن الأيمن لـ y على حاله.
  - وإلا، يقع y في الشجرة الفرعية اليمنى لـ z، ولكنه ليس الابن الأيمن لـ z (الجزء (ث)). وفي هذه الحالة، نستعيض بدايةً عن y بابنه الأيمن، ومن ثم نستعيض عن z بـ y.

ولكي نحرّك شجرات فرعية ضمن شجرة بحث ثنائية، فإننا نُعرِّف مساقًا فرعيًا "التطعيم" TRANSPLANT، الذي يستعيض عن شجرة فرعية واحدة التي هي ابن لأبيها بشجرة فرعية أخرى. عندما يستعيض TRANSPLANT عن الشجرة الفرعية التي جذرها العقدة u بالشجرة الفرعية التي جذرها v، فإن أبا العقدة u يصبح أبا العقدة v، وينتهي الأمر بأبي u بحيث يتخذ v ابنًا مناسبًا له.

الشكل 4.12 حذف عقدة z من شجرة بحث ثنائية. يمكن أن تكون z هي الجذر، أو ابنًا أيسر للعقدة q، أو ابنًا أيمن لـ q. **(أ)** ليس للعقدة z أي ابن أيسر. نستعيض عن z بابنها الأيمن r، الذي يمكن أن يساوي NIL أو لا. **(ب)** للعقدة z ابنًا أيسر l ولكن ليس لها ابن أيمن. نستعيض عن z بـ l. **(ت)** للعقدة z ابنان، الابن الأيسر هو العقدة l، والابن الأيمن هو لاحقها y، والابن الأيمن لـ y هو العقدة x. نستعيض عن z بـ y، ونُحدّث الابن الأيسر لـ y ليصبح l تاركين x كابن أيمن لـ y. **(ث)** للعقدة z ابنان (ابن أيسر l وابن أيمن r) ولاحقها y الذي لا يساوي r يقع في الشجرة الفرعية التي جذرها يقع عند r. نستعيض عن y بابنه الأيمن x، ونجعل y أبًا لـ r. ثم نجعل y ابنًا لـ q وأبًا لـ l.

```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
```

```
6  if v ≠ NIL
7      v.p = u.p
```

يعالج السطران 1 و 2 الحالة التي يكون فيها u جذر T. وإلا، فإن u هو إما ابن أيسر أو ابن أيمن لأبيه. يهتم السطران 3 و 4 بتحديث $u.p.left$ إذا كان u ابنًا أيسر، ويُحدِّث السطر 5 $u.p.right$ إذا كان u ابنًا أيمن. نسمح بأن يكون v يساوي NIL، ويُحدِّث السطران 6 و 7 $v.p$ إذا كان v لا يساوي NIL. لاحظ أن TRANSPLANT لا يحاول تحديث $v.left$ و $v.right$، إذ تقع مسؤولية القيام بذلك أو عدم القيام به على عاتق مُستدعي TRANSPLANT.

وبوجود الإجراء TRANSPLANT بين أيدينا، نورد فيما يلي الإجراء الذي يحذف عقدة z من شجرة بحث ثنائية T:

```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2      TRANSPLANT(T, z, z.right)
 3  elseif z.right == NIL
 4      TRANSPLANT(T, z, z.left)
 5  else y = TREE-MINIMUM(z.right)
 6      if y.p ≠ z
 7          TRANSPLANT(T, y, y.right)
 8          y.right = z.right
 9          y.right.p = y
10      TRANSPLANT(T, z, y)
11      y.left = z.left
12      y.left.p = y
```

يُنفِّذ الإجراء TREE-DELETE الحالات الأربع كما يلي. يُعالج السطران 1-2 الحالة التي ليس فيها لـ z ابن أيسر، ويُعالج السطران 3-4 الحالة التي يكون فيها لـ z ابن أيسر وليس له ابن أيمن. تُعالج الأسطر 5-12 الحالتين الباقيتين، واللتين لـ z فيهما ابنان. يوجد السطر 5 العقدة y، التي هي لاحق z. ولما كان لـ z شجرة فرعية يمنى غير فارغة، فيجب أن يكون لاحقه هو العقدة التي لها أصغر مفتاح في تلك الشجرة الفرعية؛ لهذا السبب جرى استدعاء TREE-MINIMUM($z.right$). وقد ذكرنا سابقًا أنه ليس لـ y ابن أيسر. نُريد أن نَنْزع y من مكانه الحالي، وأن نضعه مكان z في الشجرة. فإذا كان y الابن الأيمن لـ z، فإن الأسطر 10-12 تستعيض عن z بوصفه ابنًا لأبيه بـ y وتستعيض عن الابن الأيسر لـ y بالابن الأيسر لـ z. وإذا لم يكن y الابن الأيمن لـ z، فإن الأسطر 7-9 تستعيض عن y بوصفه ابنًا لأبيه بالابن الأيمن لـ y وتحوِّل الابن الأيمن لـ z ليكون الابنَ الأيمن لـ y، ومن ثم تستعيض الأسطر 10-12 عن z بوصفه ابنًا لأبيه بـ y وتستعيض عن الابن الأيسر لـ y بالابن الأيسر لـ z.

يستغرق تنفيذ كل سطر من TREE-DELETE، بما فيها استدعاءات TRANSPLANT زمنًا ثابتًا، ما عدا السطر 5 الذي يستدعي TREE-MINIMUM. ومن ثَمَّ، يُنفَّذ TREE-DELETE في زمن $O(h)$ على شجرة ارتفاعها h.

وبالجملة، نكون قد أثبتنا المبرهنة التالية.

مبرهنة 3.12

يمكننا تنجيز عمليات الإدراج والحذف على مجموعة ديناميكية بحيث تُنفَّذ كل منها في زمن $O(h)$ على شجرة بحث ثنائية ارتفاعها h. ■

تمارين

1-3.12

أعطِ نسخةً عوديَّة للإجراء TREE-INSERT.

2-3.12

افترض أننا بنينا شجرة بحث ثنائية بإدراج متكرِّر لقيم متمايزة في الشجرة. ناقش أن عدد العقد المُختبرَة في عملية البحث عن قيمة في الشجرة يساوي عدد العقد المُختبرَة حين إدخال القيمة أول مرة في الشجرة مضافًا إليه واحد.

3-3.12

يمكننا فرز مجموعة معطاة من n عددًا بإنشاء شجرة بحث ثنائية أولاً تحتوي هذه الأعداد (باستخدام TREE-INSERT تكراريًّا لإدراج الأعداد واحدًا تلو الآخر). ومن ثم طباعة الأعداد باستخدام إجراء تجوال في شجرة وفق الترتيب البيني. ما هو زمن التنفيذ في أسوأ الحالات وفي أحسن الحالات لخوارزمية الفرز هذه؟

4-3.12

هل عملية الحذف "تبديلية" "commutative" بمعنى أن حذف x ثم y من شجرة بحث ثنائية يُخلِّف الشجرة نفسها عند حذف y ثم x؟ ناقش لماذا نحصل على الشجرة نفسها أو أعطِ مثالاً معاكسًا يدحض ذلك.

5-3.12

افترض أنه عوضًا من أن تحتفظ العقدة x بالواصفة $x.p$، التي تشير إلى أبي x، فإنها تحتفظ بـ $x.succ$ التي تشير إلى لاحق x. أعطِ شبه رماز لتنجيز SEARSH و INSERT و DELETE على شجرة بحث ثنائية T باستخدام هذا التمثيل. علمًا بأنه يجب أن تنفَّذ الإجراءات في زمن $O(h)$، حيث h ارتفاع الشجرة T. (*تلميح*: يمكن أن ترغب بتنجيز مساقٍ فرعي يعيد أبا عقدة.)

6-3.12

يُمكننا، عندما يكون للعقدة z ابنان في TREE-DELETE، اختيار عقدة y بحيث تكون سابقتها لا لاحقتها.

ما هي التعديلات الأخرى اللازم إجراؤها على TREE-DELETE إذا قمنا بذلك. رأى البعض أن استراتيجية عادلة، متمثلة بإعطاء الأولوية نفسها للسابق واللاحق، أي إعطاء الأولوية نفسها للسابق واللاحق، تعطي نتائج تجريبية أفضل. كيف يمكن تغيير TREE-DELETE لإنجاز مثل هذه الاستراتيجية العادلة؟

★ 4.12 أشجار بحث ثنائية مبنية عشوائيًّا

بيّنا سابقًا أن كلاً من العمليات الأساسية على شجرة بحث ثنائية تنفذ في زمن $O(h)$، حيث h ارتفاع الشجرة. على أن ارتفاع شجرة بحث ثنائية يتغيَّر دومًا مع إدراج عناصر وحذفها. على سبيل المثال، إذا جرى إدراج العناصر n بترتيب متزايد تمامًا، كانت الشجرة سلسلةً بارتفاع $n-1$. من ناحية أخرى، يبين التمرين ب.5-4 أن $h \geq \lfloor \lg n \rfloor$. وكما في حالة الفرز السريع، يمكننا أن نبيّن أن سلوك الحالة الوسطى average case أكثر قربًا لأحسن الحالات منه لأسوأ الحالات.

لكننا، ولسوء الحظ، لا نعلم إلاّ القليل عن الارتفاع الوسطي لشجرة بحث ثنائية عندما يُستخدم الإدراج والحذف معًا لإنشائها. عندما تُنشأ الشجرة بالإدراج فقط، يكون التحليل قابلاً للتعقب أكثر. لذلك يمكننا أن نُعرِّف ***شجرة بحث ثنائية مبنية عشوائيًّا randomly built binary search tree*** على n مفتاحًا على أنها شجرة تنشأ من إدراج المفاتيح بترتيب عشوائي في شجرة فارغة في البداية، حيث يكون لكل من تباديل مفاتيح الدخل الـ $n!$ الاحتمال نفسه. (يُطلب إليك في التمرين 4.12-3 أن تبيّن أن هذا المفهوم مختلف عن افتراض أن كل شجرة بحث ثنائية على n مفتاحًا لها الاحتمال نفسه.) سنثبت في هذا المقطع المُبرهنةَ التالية.

مبرهنة 4.12

القيمة المتوقعة لارتفاع شجرة بحث ثنائية مبنية عشوائيًّا على n مفتاحًا تساوي $O(\lg n)$، بافتراض أن جميع المفاتيح متمايزة.

البرهان نبدأ بتعريف ثلاثة متحولات عشوائية تساعد في قياس ارتفاع شجرة بحث ثنائية مبنية عشوائيًّا. نرمز لارتفاع شجرة بحث ثنائية مبنيّة عشوائيًّا على n مفتاحًا بـ X_n، ونُعرِّف ***الارتفاع الأسي exponential height*** $Y_n = 2^{X_n}$. عندما نبني شجرة بحث ثنائية على n مفتاحًا، فإننا نختار أحد المفاتيح باعتباره جذرًا، ولنُشر بـ R_n إلى المتحول العشوائي الذي يمثّل ***مرتبة rank*** هذا المفتاح ضمن مجموعة من n مفتاحًا؛ أي إن R_n تمثّل الموقع الذي يجب أن يحتله هذا المفتاح إذا فُرِزَت مجموعةُ المفاتيح. يمكن أن تكون قيمة R_n أيَّ عنصر من المجموعة $\{1,2,\ldots,n\}$، باحتمالات متساوية. إذا كان $R_n = i$، فإن الشجرة الفرعية اليسرى للجذر هي شجرة بحث ثنائية مبنيّة عشوائيًّا على $i-1$ مفتاح، والشجرة الفرعية اليمنى للجذر هي شجرة بحث ثنائية مبنيّة عشوائيًّا على $n-i$ مفتاحًا. ولما كان ارتفاع الشجرة الثنائية أكبر بواحد من أكبر

ارتفاعي الشجرتين الفرعيتين للجذر، فإن الارتفاع الأسي للشجرة الثنائية يساوي ضعف أكبر ارتفاع أسي للشجرتين الفرعيتين للجذر. فإذا علمنا أن $R_n = i$، اقتضى ذلك أن

$$Y_n = 2 \cdot \max(Y_{i-1}, Y_{n-i}) \ .$$

وفي الحالات الأساسية، لدينا $Y_1 = 1$، لأن الارتفاع الأسي لشجرة بعقدة واحدة يساوي $2^0 = 1$، وللملاءمة نُعرف $Y_0 = 0$.

نُعرّف بعدها متحولات عشوائية مؤشرة $Z_{n,1}, Z_{n,2}, \ldots, Z_{n,n}$ حيث

$$Z_{n,i} = \mathrm{I}\{R_n = i\} \ .$$

ولما كانت قيمة R_n يمكن أن تكون أي عنصر من المجموعة $\{1, 2, \ldots, n\}$ باحتمالات متساوية، استتبع ذلك أن $\Pr\{R_n = i\} = 1/n$ في حالة $i = 1, 2, \ldots, n$، ومن ثم، يكون لدينا من التوطئة 1.5

$$\mathrm{E}[Z_{n,i}] = 1/n \ , \tag{1.12}$$

في حالة $i = 1, 2, \ldots, n$. وبالنظر إلى أن قيمة واحدة بالضبط لـ $Z_{n,i}$ تساوي 1 وجميع القيم الأخرى تساوي 0، يكون لدينا أيضًا

$$Y_n = \sum_{i=1}^{n} Z_{n,i}(2 \cdot \max(Y_{i-1}, Y_{n-i})) \ .$$

وسنبيّن أن $\mathrm{E}[Y_n]$ هو كثير حدود في n، وهذا يقتضي أخيرًا أن $\mathrm{E}[X_n] = O(\lg n)$.

ندعي أن المتحولات العشوائية المؤشرة $Z_{n,i} = \mathrm{I}\{R_n = i\}$ مستقلة عن قيم Y_{i-1} و Y_{n-i}. ولأننا اخترنا $R_n = i$، فإن الشجرة الفرعية اليسرى (التي ارتفاعها الأسي Y_{i-1}) تُبنى عشوائيًا على $i-1$ مفتاحًا التي مراتبها أقل من i. تشبه هذه الشجرةُ الفرعية تمامًا أيَّ شجرة بحث ثنائية مبنية عشوائيًا على $i-1$ مفتاحًا. وباستثناء عدد المفاتيح التي تحتويها بنيةُ هذه الشجرة الفرعية، فإنها لا تتأثر أبدًا باختيار $R_n = i$؛ ولذلك فإن المتحولين العشوائيين Y_{i-1} و $Z_{n,i}$ مستقلان. وبالمثل، فإن الشجرة الفرعية اليمنى، التي ارتفاعها الأسي Y_{n-i}، مبنية عشوائيًا على $n-i$ مفتاحًا مراتبها أكبر من i، وبنيتها مستقلة عن قيمة R_n، ومنه فإن المتحولين العشوائيين Y_{n-i} و $Z_{n,i}$ مستقلان. إذن يكون لدينا

$$\begin{aligned}
\mathrm{E}[Y_n] &= \mathrm{E}\left[\sum_{i=1}^{n} Z_{n,i}(2 \cdot \max(Y_{i-1}, Y_{n-i}))\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[Z_{n,i}(2 \cdot \max(Y_{i-1}, Y_{n-i}))] && \text{(من خطية التوقع)} \\
&= \sum_{i=1}^{n} \mathrm{E}[Z_{n,i}]\ \mathrm{E}[(2 \cdot \max(Y_{i-1}, Y_{n-i}))] && \text{(من الاستقلالية)}
\end{aligned}$$

$$= \sum_{i=1}^{n} \frac{1}{n} \mathrm{E}[(2 \cdot \max(Y_{i-1}, Y_{n-i}))] \qquad \text{(من المعادلة (1.12))}$$

$$= \frac{2}{n} \sum_{i=1}^{n} \mathrm{E}[(\max(Y_{i-1}, Y_{n-i}))] \qquad \text{(من المعادلة (ت.22))}$$

$$\leq \frac{2}{n} \sum_{i=1}^{n} (\mathrm{E}[Y_{i-1}] + \mathrm{E}[Y_{n-i}]) \qquad \text{(من التمرين (ت.3 − 4))}$$

لما كان كل حدٍّ $\mathrm{E}[Y_0], \mathrm{E}[Y_1], \ldots, \mathrm{E}[Y_{n-1}]$ يظهر مرتين في عملية الجمع الأخيرة، مرة في الحد $\mathrm{E}[Y_{i-1}]$، ومرة في الحد $\mathrm{E}[Y_{n-i}]$، فإننا نحصل على العلاقة العَوْدية

$$\mathrm{E}[Y_n] \leq \frac{4}{n} \sum_{i=0}^{n-1} \mathrm{E}[Y_i] \ . \tag{2.12}$$

وباستعمال طريقة التعويض، سنبيِّن أن للعلاقة العودية (2.12) الحل التالي، لجميع الأعداد الصحيحة الموجبة n

$$\mathrm{E}[Y_n] \leq \frac{1}{4} \binom{n+3}{3} \ .$$

وبالقيام بذلك، نستخدم المتطابقة

$$\sum_{i=0}^{n-1} \binom{i+3}{3} = \binom{n+3}{4} \ . \tag{3.12}$$

(يُطلب إليك في التمرين 12.4-1 برهان هذه المتطابقة.)

نُلاحظ، في الحالة الأساسية، أن الحدود $0 = Y_0 = \mathrm{E}[Y_0] \leq \frac{1}{4}\binom{3}{3} = 1/4$ و $1 = Y_1 = \mathrm{E}[Y_1] \leq \frac{1}{4}\binom{1+3}{3} = 1$ تبقى محققة. أما في حالة الاستقراء، فيكون لدينا

$$\mathrm{E}[Y_n] \leq \frac{4}{n} \sum_{i=0}^{n-1} \mathrm{E}[Y_i]$$

$$\leq \frac{4}{n} \sum_{i=0}^{n-1} \frac{1}{4} \binom{i+3}{3} \qquad \text{(من الفرضية الاستقرائية)}$$

$$= \frac{1}{n} \sum_{i=0}^{n-1} \binom{i+3}{3}$$

$$= \frac{1}{n} \binom{n+3}{4} \qquad \text{(من المعادلة (3.12))}$$

$$= \frac{1}{n} \cdot \frac{(n+3)!}{4!\,(n-1)!}$$

$$= \frac{1}{4} \cdot \frac{(n+3)!}{3!\,n!}$$
$$= \frac{1}{4}\binom{n+3}{3}.$$

لقد حَدّدنا $\mathrm{E}[Y_n]$، ولكن هدفنا النهائي هو أن نُحَدِّد $\mathrm{E}[X_n]$. وكما يُطلب إليك في التمرين 4-4.12 بيانه، فإن $f(x) = 2^x$ دالة مُحدَّبة (انظر المقطع ت–3 في الجزء الثاني من هذا الكتاب). لذلك، يمكننا تطبيق متراجحة جنسن Jensen's inequality (ت.26)، التي تقول بأن

$$2^{\mathrm{E}[X_n]} \leq \mathrm{E}[2^{X_n}]$$
$$= \mathrm{E}[Y_n],$$

كما يلي:

$$2^{\mathrm{E}[X_n]} \leq \frac{1}{4}\binom{n+3}{3}$$
$$= \frac{1}{4} \cdot \frac{(n+3)(n+2)(n+1)}{6}$$
$$= \frac{n^3 + 6n^2 + 11n + 6}{24}.$$

ينتج عن أخذ لغاريتم الطرفين أن $\mathrm{E}[X_n] = O(\lg n)$. ■

تمارين

1-4.12

أثبت المعادلة (3.12).

2-4.12

وصّف شجرة بحث ثنائية من n عقدة بحيث يكون العمق الوسطي لعقدة في الشجرة يساوي $\Theta(\lg n)$ في حين يكون ارتفاع الشجرة مساويًا $\omega(\lg n)$. أعطِ الحد الأعلى المقارب لارتفاع شجرة بحث ثنائية من n عقدة يكون العمق الوسطي لعقدة فيها مساويًا $\Theta(\lg n)$.

3-4.12

بيّن أن مفهوم شجرة بحث ثنائية مختارة عشوائيًّا من n مفتاحًا، حيث احتمالات اختيار كل شجرة من شجرات البحث الثنائية المؤلفة من n مفتاحًا متساوية، مختلفٌ عن مفهوم شجرة البحث الثنائية المبنية عشوائيًّا المشروحة في هذا المقطع. (*تلميح*: اسرد الاحتمالات الممكنة عندما $n = 3$.)

4-4.12

بيّن أن $f(x) = 2^x$ دالة محدبة.

$\star$ ***5-4.12***

ادرس إجراء RANDOMIZED-QUICKSORT يعمل على متتالية من n عددًا دخلاً متمايزاً. برهن، أنه مهما يكن الثابت $k > 0$، فإن جميع تباديل الدخل الـ $n!$ ما عدا $O(1/n^k)$ تُنفذ في زمن $O(n \lg n)$.

مسائل

1-12 *أشجار بحث ثنائية بمفاتيح متساوية*

تطرح المفاتيح المتساوية مشكلةً عند بناء أشجار بحث ثنائية.

أ. ما هو الأداء المقارب لـ TREE-INSERT عند استخدامها لإدراج n عنصرًا لها المفاتيح نفسها في شجرة بحث ثنائية حالتها الابتدائية فارغة؟

نقترح تحسين TREE-INSERT بإجراء اختبارٍ قبل السطر 5 لمعرفة كون $z.key = x.key$، واختبارٍ قبل السطر 11 لمعرفة كون $z.key = y.key$. نطبق إحدى الاستراتيجيات التالية في حال تحققت المساواة. أوجد، من أجل كل استراتيجية، الأداء المقارب لإدراج n عنصرًا بمفاتيح متساوية في شجرة بحث ثنائية حالتها الابتدائية فارغة. (الاستراتيجياتُ موصفةٌ لتستخدم في السطر 5، الذي نقارن فيه مفتاح z بمفتاح x. استعض عن x بـ y للوصول إلى الاستراتيجيات من أجل السطر 11.)

ب. احتفظ بالراية البوليانية $x.b$ عند العقدة x، وأسند إلى x إما $x.left$ وإما $x.right$ اعتمادًا على قيمة $x.b$، التي تتبدل بين FALSE و TRUE في كل مرة نزور فيها x خلال عملية إدراج عقدة لها مفتاح x نفسه.

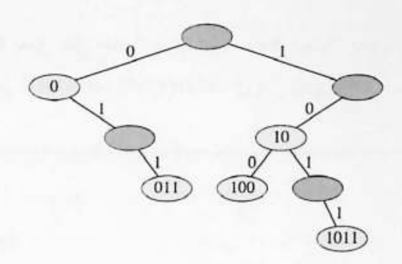
ت. احتفظ بلائحة العقد التي لها المفاتيح المتساوية عند x، وأدرج z في اللائحة.

ث. أسند إلى x القيمة $x.left$ أو $x.right$ عشوائيًا. (أعطِ الأداء في أسوأ الحالات، واستنبط زمن التنفيذ المتوقع.)

2-12 *أشجار الأسس*

ليكن لدينا سلسلتا المحارف $a = a_0a_1 \dots a_p$ و $b = b_0b_1 \dots b_q$، حيث كل من a_i و b_j هي مجموعة مرتبة من المحارف. نقول عن سلسلة المحارف a إنها ***أقل أبجديًا lexicographically less than*** من سلسلة المحارف b إذا تحقق أحد الشرطين:

1. يُوجد عدد صحيح j، حيث $0 \le j \le \min(p, q)$، وبحيث يكون $a_i = b_i$ لكل القيم $i = 0, 1, \dots, j-1$ و $a_j < b_j$، أو
2. $p < q$ و $a_i = b_i$ لكل القيم $i = 0, 1, \dots, p$.



الشكل 5.12 تُخزن شجرة أسس radix tree سلسلة محارف البتات 1011 و 10 و 011 و 100 و 0. يمكننا تحديد مفتاح كل عقدة باجتياز المسار البسيط من الجذر إلى تلك العقدة. لذلك، لا حاجة إلى تخزين المفاتيح في العقد؛ تظهر المفاتيح هنا لأغراض توضيحية فقط. تُظلَّل العقد بشدة حين لا تكون مفاتيحها موجودة في الشجرة؛ والغاية الوحيدة من وجود هذه العقد إنشاء مسار إلى عقد أخرى ليس غير.

على سبيل المثال، إذا كان a و b سلسلتي محارف بتات، فإن $10110 > 10100$ بحسب القاعدة 1 (وبجعل $j = 3$)، و $101000 > 10100$ بحسب القاعدة 2. وهذا الترتيب مشابه للترتيب المُستخدم في معاجم اللغة الإنكليزية.

تُخزِّن بنية المعطيات من نمط ***شجرة الأسس radix tree*** المبينة في الشكل 5.12 سلسلة محارف البتات 1011 و 10 و 011 و 100 و 0. ولدى البحث عن مفتاح $a = a_0a_1 \dots a_p$، فإننا نذهب يسارًا عند العقدة التي عمقها i إذا كان $a_i = 0$، ويمينًا إذا كان $a_i = 1$. لتكن S مجموعة من سلسلة بتات متمايزة مجموع أطوالها يساوي n. بيِّن كيف نستخدم شجرة الأسس لفرز S أبجديًا في زمن $\Theta(n)$. في حالة المثال في الشكل 5.12، يجب أن تكون نتيجة الفرز هي المتتالية 0, 011, 10, 100, 1011.

3-12 *العمق الوسطي لعقدة في شجرة بحث ثنائية مبنية عشوائيًا*

نبرهن في هذه المسألة أن العمق الوسطي لعقدة في شجرة بحث ثنائية مبنية عشوائيًا على n عقدة يساوي $O(\lg n)$. ومع أن هذه النتيجة أضعف من نتيجة المبرهنة 4.12، فإن الأسلوب الذي سنعتمده يُظهر تشابهًا مدهشًا بين بناء شجرة بحث ثنائية وتنفيذ الإجراء RANDOMIZED-QUICKSORT من المقطع 3.7.

نُعرِّف ***الطول الكلي للمسار total path length*** $P(T)$ لشجرة ثنائية T على أنه مجموع عمق كل العقد x في الشجرة T، ونرمز له بـ $d(x, T)$.

أ. برهن أن العمق الوسطي لعقدة في T يساوي

$$\frac{1}{n}\sum_{x \in T} d(x,T) = \frac{1}{n}P(T) .$$

وهكذا نُثبت أن القيمة المتوقعة لـ $P(T)$ تساوي $O(n \lg n)$.

ب. لنُشر بـ T_L و T_R إلى الشجرة الفرعية اليسرى والشجرة الفرعية اليمنى للشجرة T، على الترتيب. برهن أنه إذا كانت الشجرة T ذات n عقدة، فإن

$$P(T) = P(T_L) + P(T_R) + n - 1 \ .$$

ت. لنُشر بـ $P(n)$ إلى وسطي الطول الكلي للمسار لشجرة بحث ثنائية مبنية عشوائيًا على n عقدة، بيّن أن

$$P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + P(n-i-1) + n - 1) \ .$$

ث. بيّن كيف يمكن إعادة كتابة $P(n)$ كما يلي

$$P(n) = \frac{2}{n}\sum_{k=1}^{n-1} P(k) + \Theta(n) \ .$$

ج. استنتج، بالعودة إلى التحليل البديل للنسخة ذات العشوائية للفرز السريع quicksort المعطاة في المسألة 7-3، أن $P(n) = O(n \lg n)$.

نختار، عند كل استدعاء عودي لـ quicksort، عنصرًا محورياً عشوائيًا لتجزئة مجموعة العناصر الخاضعة للفرز. تُجزّئ كلُّ عقدة في شجرة البحث الثنائية مجموعةَ العناصر التي تقع في شجرة فرعية جذرها تلك العقدة.

و. وصِّف تنجيزًا للفرز السريع quicksort تكون فيه المقارنات المُستخدمة لفرز مجموعة من العناصر هي نفسها المقارنات المُستخدمة لإدراج عناصر في شجرة بحث ثنائية. (قد يختلف ترتيب هذه المقارنات، ولكن يجب إجراء المقارنات نفسها.)

4-12 عدد الأشجار الثنائية المختلفة

لنُشر بـ b_n إلى عدد الأشجار المختلفة من n عقدة. ستكتشف في هذه المسألة صيغةً لـ b_n، إضافة إلى التقدير المقارب.

أ. بيّن أن $b_0 = 1$ وأنه في حال $n \geq 1$ فإن

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k} \ .$$

ب. بالعودة إلى المسألة 4-4 لتعريف الدالة المُولّدة generating function. لتكن $B(x)$ الدالة المولّدة

$$B(x) = \sum_{n=0}^{\infty} b_n x^n \ .$$

بيّن أن $B(x) = xB(x)^2 + 1$، وبذلك تكون إحدى الطرق للتعبير عن $B(x)$ بصيغة مغلقة closed form هي

$$B(x) = \frac{1}{2x}\left(1 - \sqrt{1-4x}\right) .$$

يُعطى *نشر تايلور* ***Taylor expansion*** لـ $f(x)$ حول النقطة $x = a$ بالعلاقة

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k ,$$

حيث $f^{(k)}(x)$ هو المشتق من المرتبة k لـ f محسوبًا عند النقطة x.

ت. بيّن أن

$$b_n = \frac{1}{n+1}\binom{2n}{n}$$

(*عدد كاتالان* ***Catalan number*** من المرتبة n) باستخدام نشر تايلور لـ $\sqrt{1-4x}$ حول النقطة $x = 0$. (يمكنك أن تستعمل – بدلاً من نشر تايلور – تعميمَ نشر ثنائي الحد (ت.4) لأس غير صحيح n، حيث يمكننا كتابة $\binom{n}{k}$ لأي قيمة حقيقية n ولأي عدد صحيح k على النحو الآتي $n(n-1)\cdots(n-k+1)/k!$ حين تكون $k \geq 0$، و 0 فيما عدا ذلك.)

ث. بيّن أن

$$b_n = \frac{4^n}{\sqrt{\pi}\, n^{3/2}}\left(1 + O(1/n)\right) .$$

ملاحظات الفصل

يحتوي Knuth [211] دراسة جيدة لأشجار البحث الثنائية البسيطة إضافة إلى أشجار متنوعة عديدة. ويبدو أن اكتشاف أشجار البحث الثنائية قد تمَّ إفراديًّا على يد عدد من الأشخاص في أواخر خمسينيات القرن العشرين. وكثيرًا ما تُسمى أشجار الأسس "tries"، وقد صيغت هذه الكلمة من الحروف الوسطى لكلمة *retrieval*. وهي أيضًا مدروسة في Knuth [211].

يحتوي كثيرٌ من النصوص، ومنها أول إصدارين لهذا الكتاب، طريقةً بسيطةً نوعًا ما لخوارزمية حذف عقدة من شجرة بحث ثنائية إذا كان كلا ابنيها موجودَين. فبدلاً من الاستعاضة عن العقدة z بلاحقها y، فإننا نحذف العقدة y ولكن مع نسخ مفتاحها والمعطيات التابعة لها في العقدة z. ويتمثَّل الجانب السلبيُّ لهذا النهج في أن العقدة المحذوفة فعليًّا قد لا تكون هي العقدة المُمررة إلى إجراء الحذف. وإذا احتفظت مكونات

أخرى للبرنامج بمؤشرات إلى عقد في الشجرة، فلربما انتهى الأمر خطأ إلى مؤشراتٍ "بالية" إلى عقد جرى حذفها. ومع أن طريقة الحذف المُقدمة في هذا الإصدار من هذا الكتاب أعقد قليلاً، فإنها تضمن أن استدعاءً لحذفِ عقدة z سيحذف العقدةَ z فقط وليس غير.

يبيّن المقطع 5.15 كيف نبني شجرة بحث ثنائية أمثلية إذا عرفنا تواترات البحث قبل إنشاء الشجرة. وهذا يعني، أننا إذا علمنا تواتر البحث عن كل مفتاح وتواتر البحث عن القيم التي تقع بين المفاتيح في الشجرة، أمكننا بناء شجرة بحث ثنائية تقوم بمقتضاها مجموعةٌ للبحث تخضع لهذه التواترات بدراسة أقل عدد من العقد.

يعود الفضل في البرهان الوارد في المقطع 4.12، الذي يحدّ الارتفاع المتوقع لشجرة بحث ثنائية مبنية عشوائيًا إلى Aslam [24]. كذلك يُقدم Martınez و Roura [243] خوارزميات ذات عشوائية مضافة بغية الإدراج في شجرة بحث ثنائية والحذف منها، والتي تكون فيها نتيجة كل من العمليتين شجرة بحث ثنائية عشوائية. غير أن تعريفهما لشجرة بحث ثنائية عشوائية يختلف اختلافًا طفيفًا عن تعريف شجرة بحث ثنائية مبنية عشوائيًا، الوارد في هذا الفصل.

13 الأشجار الحمراء–السوداء

رأينا في الفصل 12 أنه يمكن لشجرة بحث ثنائية ذات ارتفاع h أن تدعم أيًّا من عمليات المجموعات الديناميكية الأساسية – مثل البحث SEARCH والسَّابق PREDECESSOR والخَلَف SUCCESSOR والحد الأدنى MINIMUM والحد الأعلى MAXIMUM والإدراج INSERT والحذف DELETE — في زمن $O(h)$. لذلك، تكون عمليات المجموعات سريعة إذا كان ارتفاع شجرة البحث صغيرًا. فإذا كان ارتفاعها كبيرًا فقد لا تُنفَّذ هذه العمليات بسرعة أكبر مما لو استُخدمت قائمة مرتبطة. والأشجار الحمراء–السوداء هي أحد أشكالٍ كثيرةٍ لأشجار البحث التي جُعلت "متوازنة" لكي تضمن أن تستغرق عمليات المجموعات الديناميكية الأساسية في أسوأ الحالات زمنًا $O(\lg n)$.

1.13 خصائص الأشجار الحمراء–السوداء

الشجرة الحمراء–السوداء red-black tree هي شجرة بحث ثنائية تتضمن بت تخزين إضافي واحد في كل عقدة من عقدها، يعبِّر عن لون العقدة الذي يمكن أن يكون أحمر أو أسود. وبتقييد ألوان العقد على أيِّ مسار بسيط، من الجذر إلى إحدى الأوراق، تضمن الأشجار الحمراء–السوداء أنه لا يوجد مسارٌ يتجاوز طولُه ضعفَيْ طول أي مسارٍ آخر، أي إن الشجرة ***متوازنة balanced*** تقريبًا.

أصبحت كل عقدة من الشجرة تتضمن الواصفات attributes التالية: اللون $color$ والمفتاح key والابن الأيمن $right$ والابن الأيسر $left$ والأب p. فإذا لم يكن الأب أو أحد الأبناء موجودًا تكون قيمة حقل المؤشر الموافق للعقدة معدومة NIL. وسننظر إلى هذه المؤشرات المعدومة (NIL) على أنها مؤشرات إلى أوراق (عقد خارجية) في الشجرة الثنائية، وإلى العقد العادية الحاملة للمفاتيح على أنها عقد داخلية في الشجرة.

الشجرة الحمراء–السوداء هي شجرة بحث ثنائية تحقق ***الخصائص الحمراء–السوداء red-black properties*** التالية:

1. كل عقدة إما أن تكون حمراء أو سوداء.
2. عقدة الجذر سوداء.

3. كل ورقة (NIL) هي سوداء.

4. إذا كانت إحدى العقد حمراء، كانت عقدتا أبنائها سوداوَيْن.

5. كل المسارات البسيطة من أيِّ عقدةٍ إلى الأوراق النازلة منها تحتوي العدد نفسه من العقد السوداء.

يُظهر الشكل 1.13(أ) مثالاً على شجرة حمراء-سوداء.

نستخدم حارسًا وحيدًا لتمثيل القيم المعدومة NIL لتسهيل التعامل مع الشروط الحدِّية في رماز الأشجار الحمراء-السوداء (انظر الصفحة 239). ففي شجرة حمراء-سوداء T يكون الحارس $T.nil$ غرضًا له نفس واصفات أي عقدة عادية في الشجرة. ويكون واصف *اللون* *color* فيه أسود BLACK، وأما بقية الحقول – *الأب* p *والابن الأيسر* $left$ *والابن الأيمن* $right$ *والمفتاح* key – فيمكن أن تأخذ قيمًا اعتباطية. وكما يُظهر الشكل 1.13(ب) فإنَّ كل المؤشرات إلى NIL يستعاض عنها بمؤشرات إلى الحارس $T.nil$.

نستخدم الحارس بحيث يمكننا معالجة ابن معدوم NIL لعقدة ما x كعقدة عادية أبوها x. ومع أنه يمكننا بدل ذلك أن نضيف عقدة حارس متمايزة لكل عقدة معدومة NIL في الشجرة، وبحيث يكون الأب الخاص بكل قيمة معدومة NIL معرفًا تعريفًا جيدًا، فإن ذلك النهج سيبدد الحجوم. بدلاً من ذلك نستخدم الحارس الوحيد $T.nil$ لتمثيل جميع القيم المعدومة NIL — أي كل الأوراق والأب الخاص بالجذر. إن قيم واصفات الحارس p و $left$ و $right$ و key ليست ذات أهمية، ومع ذلك يمكننا – للتسهيل – أن نعطيها قيمًا اصطلاحية خلال سير إجراء معين.

نوجه اهتمامنا عمومًا إلى العقد الداخلية من الشجرة الحمراء-السوداء لأنها تحمل قيم المفاتيح. سنُغفل فيما تبقى من هذا الفصل الأوراق عند رسم شجرة حمراء-سوداء، كما يبين الشكل 1.13(ت).

نسمي عدد العقد السوداء على أي مسار بسيط من عقدة ما x (غير متضمنة) نزولاً إلى ورقة ما **الارتفاع الأسود** ***black-height*** للعقدة، ونرمز له بـ $\text{bh}(x)$. وحسب الخاصية رقم 5، فإنَّ مفهوم الارتفاع الأسود معرَّف جيدًا، لأن كل المسارات البسيطة النازلة من العقدة لها العدد نفسه من العقد السوداء. نعرِّف الارتفاع الأسود لشجرة حمراء-سوداء على أنه الارتفاع الأسود لجذرها.

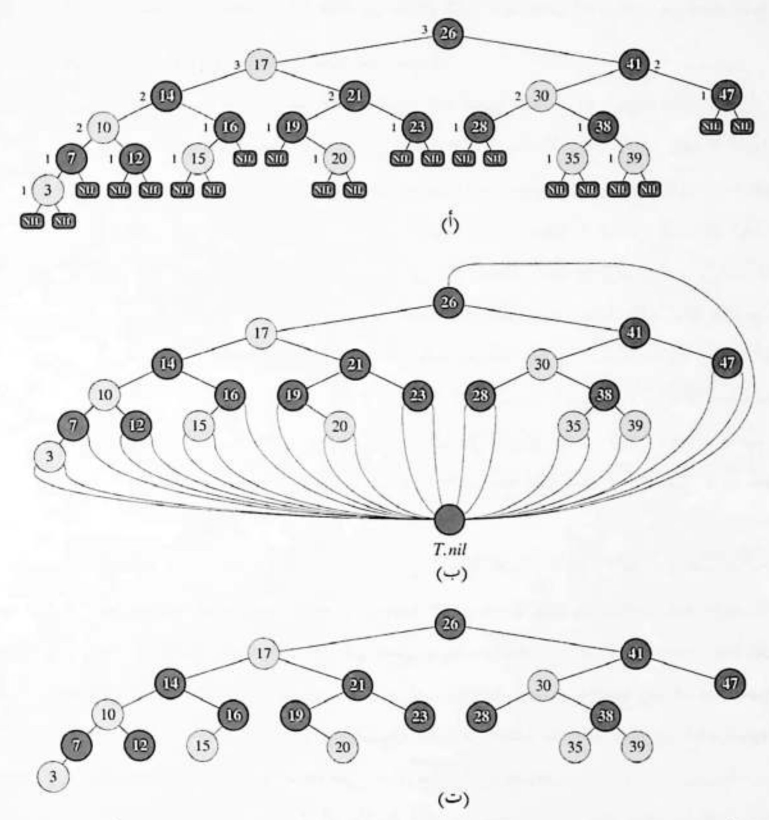
تُظهر التوطئة التالية السبب في أن الأشجار الحمراء-السوداء أشجار بحث جيدة.

توطئة 1.13

كل شجرة حمراء-سوداء ذات n عقدة داخلية يكون ارتفاعها على الأكثر $2\lg(n+1)$

البرهان نبدأ ببيان أن الشجرة الفرعية التي جذرها عند أية عقدة x تتضمن على الأقل $2^{\text{bh}(x)}-1$ عقدة داخلية. نبرهن على هذا بالاستقراء على ارتفاع x. إذا كان ارتفاع x هو 0، فإنَّ x يجب أن يكون ورقة ($T.nil$)، وتكون الشجرة الفرعية التي جذرها x تتضمن فعليًّا $2^{\text{bh}(x)}-1=2^0-1=0$ عقدة داخلية

على الأقل. نفترض لخطوة الاستقراء عقدة ما x ذات ارتفاع موجب، وهي عقدة داخلية لها ابنان. كل ابن له ارتفاع أسود مقداره $\text{bh}(x)$ أو $\text{bh}(x)-1$، وذلك تبعًا للونه (أحمر أو أسود على الترتيب). ولما كان ارتفاع



الشكل 1.13 شجرة حمراء-سوداء تظهر فيها العقد السوداء بلون غامق والحمراء مظللة. كلُّ عقدة في الشجرة الحمراء-السوداء إما أن تكون حمراء أو سوداء، وابنا أي عقدة حمراء سوداوان، وكل مسار بسيط من عقدة ما إلى الأوراق النازلة منها يحتوي العدد نفسه من العقد السوداء. (أ) كل ورقة تظهر بأنها معدومة NIL تكون سوداء. كل عقدة غير معدومة تكون معلَّمة بارتفاعها الأسود؛ والارتفاع الأسود للعقد المعدومة هو 0. **(ب)** الشجرة الحمراء-السوداء نفسها لكن مع الاستعاضة عن كل عقدة معدومة NIL بالحارس الوحيد $T.nil$ الذي يكون أسود دومًا، ومع إغفال الارتفاعات السوداء. العقدة الأب للجذر هي الحارس كذلك. **(ت)** الشجرة الحمراء-السوداء نفسها لكن مع إغفال الأوراق والأب الخاص بالجذر كليًا. سنستخدم هذا الأسلوب في الرسم حتى نهاية هذا الفصل.

أحد أبناء x أقل من ارتفاع x نفسها، فيمكننا تطبيق الفرضية الاستقرائية لنستنتج أنَّ كل ابن لديه على الأقل $2^{\text{bh}(x)-1} - 1$ عقدة داخلية. إذن فالشجرة الفرعية ذات الجذر x تتضمن على الأقل $(2^{\text{bh}(x)-1} - 1) + (2^{\text{bh}(x)-1} - 1) + 1 = 2^{\text{bh}(x)} - 1$ عقدة داخلية، وهو المطلوب.

ولاستكمال برهان التوطئة، ليكن h ارتفاع الشجرة. فبحسب الخاصية 4، يجب أن يكون على الأقل نصف العقد الواقعة على أي مسار بسيط من الجذر إلى الأوراق، باستثناء الجذر، سوداء. وهذا يستتبع أن الارتفاع الأسود للجذر يجب أن يكون على الأقل $h/2$، ومن ثمَّ

$$n \geq 2^{h/2} - 1 \ .$$

بنقل الواحد إلى الجانب الأيسر، وأخذ لغاريتم الطرفين يكون لدينا $\lg(n+1) \geq h/2$ أو $h \leq 2\lg(n+1)$. ■

ينتج مباشرةً عن هذه التوطئة أنَّنا يمكن أن ننجِّز عمليات المجموعات الديناميكية: SEARCH، MINUMUM، MAXIMUM، SUCCESSOR و PREDECESSOR في زمن $O(\lg n)$ على الأشجار الحمراء-السوداء، لأن كلاً منها يمكن أن تُنفَّذ في زمن $O(h)$ على شجرة بحث ثنائية ذات ارتفاع h (كما تقدَّم في الفصل 12)، وأي شجرة حمراء-سوداء ذات n عقدة هي شجرة بحث ثنائية بارتفاع $O(\lg n)$. (يجب طبعًا أن تُستبدل بكل مواضع ورود NIL في خوارزميات الفصل 12 كلمة $T.nil$.) ومع أنَّ الخوارزميات TREE-INSERT وTREE-DELETE في الفصل 12 تنفَّذ في زمن $O(\lg n)$ عندما يكون الدخل شجرة حمراء-سوداء، إلا أنها لا تدعم مباشرةً عمليات المجموعات الديناميكية: INSERT و DELETE لأنها لا تضمن أن تكون شجرةُ البحث الثنائية المعدَّلة شجرةً حمراء-سوداء. مع ذلك، سنرى في المقطع 3.13 والمقطع 4.13 كيف يمكننا تنفيذ هاتين العمليتين في زمن $O(\lg n)$.

تمارين

1-1.13

على غرار الشكل 1.13(أ)، ارسم شجرة البحث الثنائية الكاملة ذات الارتفاع 3، والمفاتيح $\{1, 2, \ldots, 15\}$. أضف الأوراق NIL ولوِّن العقد بثلاث طرق مختلفة، بحيث تكون الارتفاعات السوداء للأشجار الحمراء-السوداء الناتجة هي 2 و 3 و 4.

2-1.13

ارسم الشجرة الحمراء-السوداء التي تنتج بعد استدعاء TREE-INSERT على الشجرة التي في الشكل 1.13 مع المفتاح 36. وإذا كانت العقدة المدرجة ملوَّنة بالأحمر، فهل الشجرة الناتجة هي شجرة حمراء-سوداء؟ ماذا لو كانت ملوَّنة بالأسود؟

3-1.13

لنعرِّف ***شجرة حمراء-سوداء مرخاة relaxed red-black tree*** على أنها شجرة بحث ثنائي تحقِّق خصائص الأشجار الحمراء-السوداء 1 و 3 و 4 و 5. وبتعبير آخر، يمكن أن يكون الجذر أحمر أو أسود. لنأخذ شجرة T من هذا النوع جذرها أحمر. إذا لوَّنّا جذر T بالأسود ولم نجرِ أي تغيير آخر على T، فهل تكون الشجرة الناتجة شجرة حمراء-سوداء؟

4-1.13

افترض أننا "نمتص" كل عقدة حمراء في شجرة حمراء-سوداء في أمها السوداء، بحيث يصبح أبناء العقدة الحمراء أبناء للعقدة الأم السوداء. (تجاهل ما يحدث للمفاتيح.) ما هي الدرجات الممكنة لعقدة سوداء بعد أن يجري امتصاص جميع أبنائها الحمر؟ ماذا تقول عن أعماق الأوراق في الشجرة الناتجة؟

5-1.13

بيِّن أن أطول مسار بسيط من عقدة x في شجرة حمراء-سوداء إلى ورقة نازلة منها، يبلغ طوله على الأكثر ضعفي طول أقصر مسار بسيط من عقدة x إلى ورقة نازلة منها.

6-1.13

ما هو العدد الأعظمي المحتمل للعقد الداخلية في شجرة حمراء-سوداء ارتفاعها الأسود k؟ وما هو العدد الأصغري المحتمل؟

7-1.13

صِفْ شجرةً حمراء-سوداء تتضمن n مفتاح تكون فيها نسبة العقد الداخلية الحمراء إلى العقد الداخلية السوداء أعلى ما يمكن. وما هي هذه النسبة؟ ما هي الشجرة التي يكون لها أدنى نسبة محتملة؟ وما هي هذه النسبة؟

2.13 الدورانات

تستغرق عمليات أشجار البحث TREE-INSERT و TREE-DELETE زمنًا $O(\lg n)$ عند تنفيذها على شجرة حمراء-سوداء تتضمن n مفتاحًا. ولما كانت هذه العمليات تعدِّل الشجرة فإنَّ النتيجة يمكن أن تخرق خصائص الأشجار الحمراء-السوداء المذكورة في المقطع 1.13. ولإعادة تحقيق هذه الخصائص، يجب أن نُغيِّر ألوان بعض العقد في الشجرة وأن نُغيِّر بنية المؤشر.

نُغيِّر بنية المؤشِّر ***بالدوران rotation***، وهي عملية محلية في شجرة بحث تحافظ على خاصية شجرة البحث الثنائية. يُظهر الشكل 2.13 نوعَي الدورانات: الدورانات إلى اليسار والدورانات إلى اليمين. عندما نُجري دورانًا إلى اليسار على عقدة x، نفترض أن ابنها الأيمن y ليس معدومًا $T.nil$؛ ويمكن أن تكون x أية عقدة في



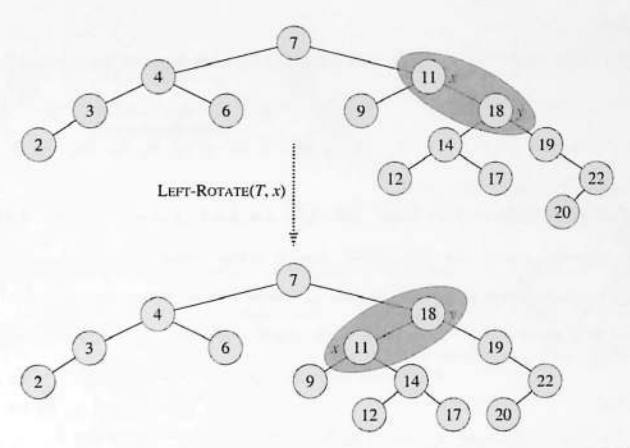
الشكل 2.13 عمليات الدوران في شجرة بحث ثنائية. تحوّل العمليةُ LEFT-ROTATE(T,x) تشكيلةَ العقدتين الموجودة إلى يمين الشكل إلى التشكيلة الموجودة إلى يسار الشكل بتغيير عدد ثابت من المؤشرات. تحوّل العملية المعاكسة RIGHT-ROTATE(T,y) التشكيلة الموجودة إلى اليسار إلى التشكيلة الموجودة إلى اليمين. وتمثّل الأحرف α و β و γ أشجارًا فرعية اعتباطية. تحفظ عملية الدوران خاصية شجرة البحث الثنائية: المفاتيح في α تسبق $x.key$، التي تسبق المفاتيح في β، التي تسبق $y.key$، التي تسبق بدورها المفاتيح في γ.

الشجرة ابنها الأيمن غير معدوم $T.nil$. الدوران الأيسر "يرتكز" على الرابط من x إلى y. ويجعل من y الجذرَ الجديدَ للشجرة الفرعية، ويصبح x الابن الأيسر لـ y، ويصبح الابن الأيسر لـ y هو الابن الأيمن لـ x.

يَفترض شبه رماز LEFT-ROTATE أنَّ $x.right \neq T.nil$ وأنَّ أبا الجذر هو $T.nil$.

```
LEFT-ROTATE(T, x)
 1  y = x.right              // set y
 2  x.right = y.left // turn y's left subtree into x's right subtree
 3  if y.left ≠ T.nil
 4      y.left.p = x
 5  y.p = x.p                // link x's parent to y
 6  if x.p == T.nil
 7      T.root = y
 8  elseif x == x.p.left
 9      x.p.left = y
10  else x.p.right = y
11  y.left = x               // put x on y's left
12  x.p = y
```

يُظهر الشكل 3.13 مثالاً عن كيفية تعديل الإجراء LEFT-ROTATE شجرةَ بحثٍ ثنائية. رماز RIGHT-ROTATE هو نظير للرماز السابق. يعمل كل من LEFT-ROTATE وRIGHT-ROTATE في زمن $O(1)$. المؤشرات فقط هي التي تتغير في الدوران؛ وتبقى الواصفات الأخرى في العقدة كما هي.



الشكل 3.13 مثال عن كيفية تعديل الإجراء LEFT-ROTATE(T, x) شجرةً بحث ثنائية. ينتج عن المسير بالترتيب الداخلي في كلٍّ من شجرة الدخل والشجرة المعدَّلة القائمة نفسها من المفاتيح.

تمارين

1-2.13

اكتب شبه رماز العملية RIGHT-ROTATE.

2-2.13

برهن أن في كلِّ شجرةِ بحثٍ ثنائية ذات n عقدة $n-1$ دورانًا ممكنًا تمامًا.

3-2.13

لتكن a و b و c عقدًا اعتباطية في الأشجار الفرعية α و β و γ على الترتيب في الشجرة اليمنى من الشكل 2.13. كيف يتغير عمق كل من a و b و c عند إجراء دوران نحو اليسار على العقدة x المبينة في الشكل؟

4-2.13

بيِّن أنَّ أي شجرة بحث ثنائية اعتباطية ذات n عقدة يمكن أن تُحوَّل إلى أية شجرة بحث ثنائية اعتباطية أخرى ذات n عقدة باستخدام $O(n)$ دورانًا. (*تلميح*: بيِّن أولاً أنَّه يكفي $n-1$ دورانًا إلى اليمين على الأكثر لتحويل الشجرة إلى سلسلة تبدأ من اليمين.)

★ ***5-2.13***

نقول إن شجرة بحث ثنائية T_1 يمكن أن ***تحوَّل يمينًا right-converted*** إلى شجرة بحث ثنائية T_2 إذا أمكن الحصول على T_2 من T_1 عبر سلسلة من استدعاءات لـ RIGHT-ROTATE. أعطِ مثالاً عن شجرتين T_1 و T_2 بحيث لا يمكن لـ T_1 أن تحوَّل يمينًا إلى T_2. ثم بيِّن أنه إذا أمكن تحويل شجرة T_1 يمينًا إلى T_2، فإنَّ هذا التحويل يمكن أن يجري باستخدام $O(n^2)$ استدعاء لـ RIGHT-ROTATE.

3.13 الإدراج

يمكننا إدراج عقدة في شجرة حمراء-سوداء ذات n عقدة في زمن $O(\lg n)$. ولإجراء ذلك نستخدم نسخة معدَّلةً قليلاً من إجراء TREE-INSERT (المقطع 3.12) لإدراج عقدة z في شجرة T كما لو كانت شجرة بحث ثنائية عادية، ثمَّ نُلوِّن z بالأحمر. (يُطلب إليك في التمرين 3.13-1 أن تفسِّر لِمَ اخترنا تلوين العقدة z بالأحمر بدلاً عن الأسود.) ولضمان الحفاظ على الخصائص الحمراء-السوداء، نستدعي إجراءً مساعدًا RB-INSERT-FIXUP لإعادة تلوين العقد وإجراء الدورانات. إنَّ استدعاء RB-INSERT(T,z) يدرج عقدة z، يُفترض أن مفتاحها قد مُلِئَ سابقًا، ضمن الشجرة الحمراء-السوداء T.

```
RB-INSERT(T, z)
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == T.nil
10      T.root = z
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

هناك أربعة اختلافات بين الإجراءين TREE-INSERT وRB-INSERT. أولاً، كل القيم NIL في TREE-INSERT جرى الاستعاضة عنها بـ $T.nil$. ثانيًا، نجعل قيمة كل من $z.left$ و $z.right$ مساوية $T.nil$ في الأسطر 14-15 من RB-INSERT، للحفاظ على بنية الشجرة. ثالثًا، نُلوِّن z بالأحمر في السطر 16. رابعًا، لما كان تلوين z بالأحمر يمكن أن يسبِّب خرقًا لإحدى الخصائص الحمراء-السوداء، فإننا نستدعي RB-INSERT-FIXUP(T,z) في السطر 17 من RB-INSERT لاستعادة الخصائص الحمراء-السوداء.

```
RB-INSERT-FIXUP(T, z)
 1  while z.p.color == RED
 2      if z.p == z.p.p.left
 3          y = z.p.p.right
 4          if y.color == RED
```

```
5            z.p.color = BLACK                          // Case 1
6            y.color = BLACK                            // Case 1
7            z.p.p.color = RED                          // Case 1
8            z = z.p.p                                  // Case 1
9        else if z == z.p.right
10               z = z.p                                // Case 2
11               LEFT-ROTATE(T, z)                      // Case 2
12           z.p.color = BLACK                          // Case 3
13           z.p.p.color = RED                          // Case 3
14           RIGHT-ROTATE(T, z.p.p)                     // Case 3
15       else (same as then clause with "right" and "left" exchanged)
16   T.root.color = BLACK
```

ولمعرفة كيفية عمل الإجراء RB-INSERT-FIXUP سنجزّئ دراستنا للرماز إلى ثلاث خطوات رئيسية. أولاً، سنُحدِّد ما هي الخروقات للخصائص الحمراء-السوداء التي حصلت في RB-INSERT عند إدراج العقدة z وتلوينها بالأحمر. ثانيًا، سنتفحص الهدف العام لحلقة **while** في الأسطر 1-15. أخيرًا، سنستعرض كلاً من الحالات الثلاث[1] ضمن جسم حلقة **while** ونرى كيف حقَّقت الهدف. يُظهر الشكل 4.13 كيف يعمل RB-INSERT-FIXUP على عينة شجرة حمراء-سوداء.

أيٌّ من الخصائص الحمراء-السوداء يُمكن أن تُخرق بعد استدعاء RB-INSERT-FIXUP؟ الخاصية 1 تبقى صحيحة بالتأكيد، وكذلك الخاصية 3، لأن كِلا ابني العقدة الحمراء الجديدة المدرجة حديثًا هما الحارس $T.nil$. أما الخاصية 5 التي تعني أنَّ عدد العقد السوداء هو نفسه في جميع المسارات البسيطة من عقدة معينة، هي أيضًا محقَّقة، لأن العقدة z تستبدل الحارس (الأسود)، والعقدة z حمراء ولديها أبناء حراس. لذلك فإنَّ الخاصيتين اللتين يمكن أن يجري خرقهما هما الخاصية 2 التي تتطلب أن يكون الجذر أسود، والخاصية 4 التي تعني أنَّه لا يمكن لعقدة حمراء أن يكون لها ابن أحمر. وكلا الخرقَيْن ينتجان عن تلوين z بالأحمر. تُخْرَق الخاصية 2 إذا كان z هو الجذر، وتُخْرَق الخاصية 4 إذا كان أبو z أحمر. يُظهر الشكل 4.13(أ) خَرْقَ الخاصية 4 بعد إدراج العقدة z.

تحتفظ حلقة **while** في الأسطر 1-15 باللامتغير الثلاثي الأجزاء الآتي في بداية كل تكرار من الحلقة:

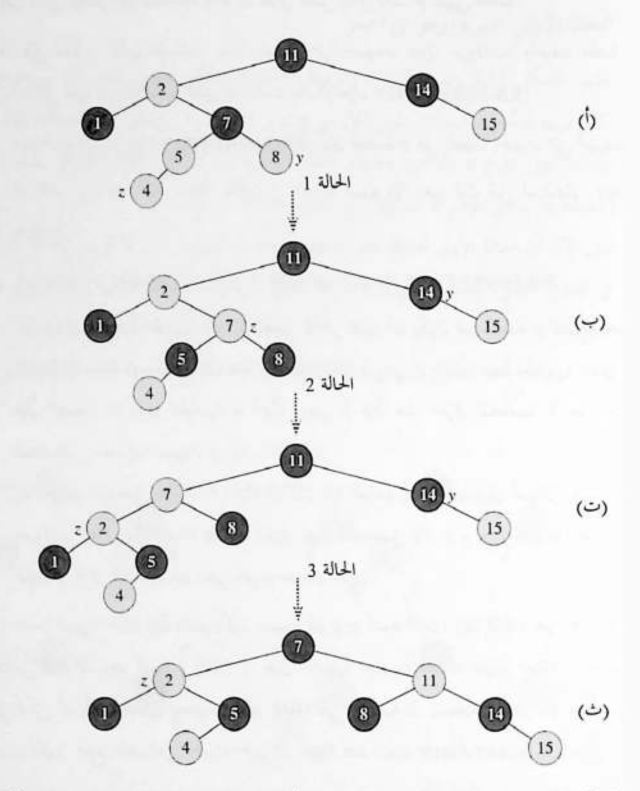
أ. العقدة z حمراء.

ب. إذا كان $z.p$ هو الجذر، فإنه يكون أسود.

ت. إذا خرقت الشجرة أيًا من الخصائص الحمراء-السوداء، فستخرق واحدة منها على الأكثر، وسيكوين الخرق إما للخاصية 2 أو الخاصية 4. فإذا خَرقت الشجرةُ الخاصيةَ 2 فسبب ذلك أن z هو الجذر وهو أحمر. وإذا خَرقت الشجرة الخاصيةَ 4، فسبب ذلك أن لون كلٍّ من z و $z.p$ أحمر.

[1] الحالة 2 تنتهي إلى الحالة 3، أي أنَّ هاتين الحالتين لا تقصي إحداهما الأخرى.

الجزء (ت) الذي يتناول خرقَ الخصائص الحمراء-السوداء، هو الجزء الأكثر أهميةً لبيان استعادة هذه الخصائص في RB-INSERT-FIXUP والذي نستخدمه دومًا لفهم الحالات ضمن الرماز. ولمّا كنا نركز على العقدة z والعقد التي بجوارها في الشجرة، فمن المفيد أن نعرف من الجزء (أ) أن z لونها أحمر. سنستخدم الجزء (ب) لإظهار وجود العقدة $z.p.p$ عندما نشير إليها في الأسطر 2 و 3 و 7 و 8 و 13 و 14.



الشكل 4.13 عملية RB-INSERT-FIXUP. (أ) عقدة z بعد الإدراج. لما كانت z حمراء وأبوها $z.p$ كذلك، فإنه يَحدث خرقٌ للخاصية 4. ولما كان عم z وهو y أحمر، فإن الحالة الأولى من الرماز case 1 تنطبق. نعيد تلوين العقد وننقل المؤشر z إلى الأعلى ضمن الشجرة، فتنتج الشجرة الظاهرة في (ب). مرةً أخرى، z وأبوها كلاهما حمراوان، لكن عم z وهو y أسود. ولما كانت z هي الابن الأيمن لـ $z.p$، فإن الحالة الثانية case 2 تنطبق. نجري دورانًا إلى اليسار، فتَظهر الشجرة الناتجة في الشكل (ت). أصبح الآن z هو الابن الأيسر لأبيه، فتنطبق الحالة الثالثة case 3. بإعادة التلوين وتطبيق دوران إلى اليمين تنتج الشجرة في (ث)، وهي شجرة حمراء-سوداء صحيحة.

تذكَّر أننا نحتاج إلى إظهار أنَّ لامتغير الحلقة صحيح قبل التكرار الأول للحلقة، وأن كل تكرار يحافظ على لامتغير الحلقة، وأن لامتغير الحلقة يعطينا خاصية مفيدة في نهاية الحلقة.

نبدأ بمناقشة الاستبداء والانتهاء. ثم، وفي سياق دراستنا لكيفية عمل الحلقة بتفصيل أكبر، سنبرهن أن الحلقة تحافظ على اللامتغير في كل تكرار. سنبرهن أيضًا أن كل تكرار من الحلقة له نتيجتان محتملتان: إما أن ينتقل المؤشر z إلى الأعلى في الشجرة، وإما أن نجري بعض الدورانات ثم تنتهي الحلقة.

الاستبداء: قبل التكرار الأول للحلقة، بدأنا بشجرة حمراء-سوداء بدون خروقات، وأضفنا عقدة حمراء z. نبيِّن أنَّ كل جزء من اللامتغير محقق عند استدعاء الإجراء RB-INSERT-FIXUP:

أ. عند استدعاء الإجراء RB-INSERT-FIXUP، فإن العقدة z هي العقدة الحمراء التي أضيفت.

ب. إذا كان $z.p$ هو الجذر، فإنَّه يكون في البداية أسود ولا يتغير لونه قبل استدعاء RB-INSERT-FIXUP.

ت. رأينا سابقًا أنَّ الخصائص 1 و 3 و 5 محققة عند استدعاء RB-INSERT-FIXUP.

إذا خرقت الشجرةُ الخاصيةَ 2، فإنَّ الجذر الأحمر يجب أن يكون هو العقدة z المضافة حديثًا، وهي العقدة الداخلية الوحيدة في الشجرة. ولما كان لكلٍّ من أبي z وابنَيْها قيمةُ الحارس، الذي هو أسود، فإنَّ الشجرة لا تخرق الخاصية 4 أيضًا. ومن ثَمَّ فإنَّ هذا الخرق للخاصية 2 هو الخرق الوحيد للخصائص الحمراء-السوداء في كامل الشجرة.

إذا خرقت الشجرة الخاصية 4، فإنَّه لما كان ابنا العقدة z هما حارسَيْن أسودَيْن وليس في الشجرة خروقات أخرى قبل إضافة z، فإنَّ الخرق يجب أن يحصل لأن z و $z.p$ كليهما أحمران. وفيما عدا ذلك لا تَخرق الشجرةُ خصائصَ حمراء-سوداء أخرى.

الانتهاء: عندما تنتهي الحلقة فإنَّ ذلك يكون بسبب أن $z.p$ أسود اللون. (إذا كان z هو الجذر فإنَّ $z.p$ هو الحارس $T.nil$، وهو أسود.) لذلك لا تَخرق الشجرةُ الخاصيةَ 4 عند انتهاء الحلقة. والخاصية الوحيدة التي يمكن أن يُخفق تحقُّقها بوجود لامتغير الحلقة هي الخاصية 2. يستعيد السطر 16 هذه الخاصية أيضًا بحيث تكون جميع الخصائص الحمراء-السوداء محققة عند انتهاء RB-INSERT-FIXUP.

المحافظة: نحتاج فعليًا لاعتبار ست حالات في حلقة **while**، غير أنَّ ثلاثًا منها مناظرةٌ للثلاث الأخرى، بحسب ما يتحدَّد في السطر 2 من كون أبو z وهو $z.p$ ابنًا أيسر أو ابنًا أيمن لجد z وهو $z.p.p$. وقد أعطينا فقط الرماز الخاص بالحالة التي يكون فيها $z.p$ ابنًا أيسر. إن العقدة $z.p.p$ موجودة لأنه بحسب الجزء (ب) من لامتغير الحلقة، إذا كان $z.p$ هو الجذر فإنه يكون أسود. ولما كنا ندخل إلى تكرار الحلقة فقط إذا كانت $z.p$ حمراء، فإننا نعلم أنَّ $z.p$ لا يمكن أن تكون هي الجذر، ومن ثمَّ فإنَّ $z.p.p$ موجودة.

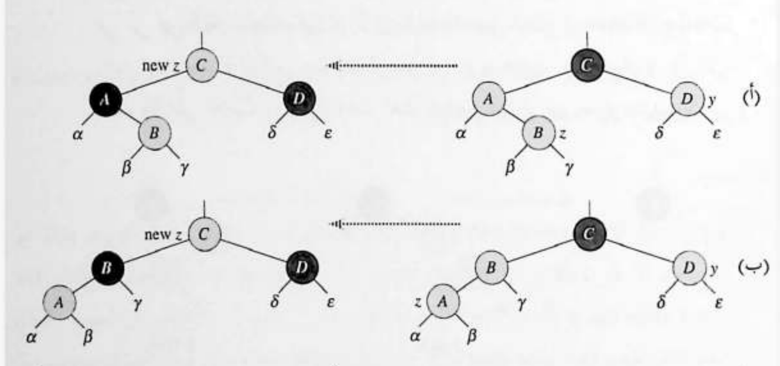
نميز الحالة الأولى case 1 عن الحالتين 2 و 3 بلون ذرِّية الأب العائد لـ z أو "العم". إن السطر 3 يجعل العقدة y تشير إلى عم z وهو $z.p.p.right$، في حين يختبر السطر 4 لون y. فإذا كانت y حمراء نُنفَّذ الحالة 1، وإلا ينتقل التحكم إلى الحالتين 2 و3. في جميع الحالات الثلاث، يكون جد z وهو $z.p.p$ أسود، لأن أباه $z.p$ لونه أحمر، والخاصية 4 تُخترَق بين z و $z.p$ فقط.

الحالة الأولى: عم z (وهو y) أحمر

يُظهر الشكل 5.13 وضع الحالة 1 (الأسطر 5-8) التي تُنفَّذ عندما يكون كلٌّ من $z.p$ و y حمراوين. ولما كان $z.p.p$ أسود، فيمكننا تلوين كلٍّ من $z.p$ و y بالأسود، وبذلك نحل مشكلة كون z و $z.p$ حمراوَيْن، ويمكننا تلوين $z.p.p$ بالأحمر، وبذلك نحفظ الخاصية 5. ثم نكرر الحلقة **while** باعتبار $z.p.p$ هو العقدة الجديدة z. ينتقل المؤشر z مستويين إلى أعلى الشجرة.

نبين الآن أن الحالة الأولى تُحافظ على لامتغير الحلقة في بداية التكرار التالي. نستخدم z للتعبير عن العقدة z في التكرار الحالي، ونستخدم $z' = z.p.p$ للتعبير عن العقدة التي ستسمى z في اختبار السطر 1 في التكرار التالي.

أ. لمّا كان هذا التكرار يلوِّن $z.p.p$ بالأحمر، فإنَّ z' تكون حمراء في بداية التكرار التالي.



الشكل 5.13 الحالة الأولى من الإجراء RB-INSERT-FIXUP. تُخرَق الخاصية 4 لأن z وأباها $z.p$ حمراوان. يُنفَّذ العمل نفسه سواءٌ أكان (أ) z ابنًا أيمن أو (ب) z هو ابنًا أيسر. لكلٍّ من الأشجار الفرعية α و β و γ و δ و ε جذر أسود، ولكل منها الارتفاع الأسود نفسه. يغيِّر الرماز في الحالة الأولى ألوان بعض العقد، محافظًا على الخاصية 5: جميع المسارات البسيطة النازلة من عقدة إلى ورقة تحتوي العدد نفسه من العقد السوداء. تتابع حلقة **while** مع العقدة الجد لـ z وهي $z.p.p$ باعتبارها عقدة z جديدة. يمكن أن يحدث الآن أي خرق للخاصية 4 فقط بين العقدة الجديدة z (الحمراء) وأبيها، إذا كان أحمر أيضًا.

ب. العقدة $z'.p$ هي $z.p.p.p$ في هذا التكرار، ولونها لا يتغيَّر. فإذا كانت هي الجذر فلونها كان أسود قبل هذا التكرار، ويبقى كذلك في بداية التكرار الذي يليه.

ت. سبق أن بينًا أنَّ الحالة الأولى تحافظ على الخاصية 5، ولا تُحدث خرقًا للخاصتين 1 أو 3 .

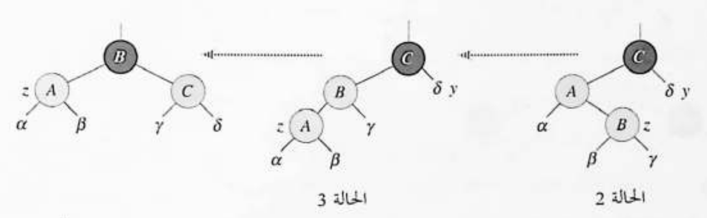
إذا كان z' هو الجذر في بداية التكرار التالي، فتكون الحالة الأولى قد صححت الخرق الوحيد للخاصية 4 في هذا التكرار. ولأن z' لونها أحمر وهي الجذر، فإنَّ الخاصية 2 تصبح هي الخاصية الوحيدة المخروقة ويعود سبب خرقها إلى z'.

إذا لم يكن z' هو الجذر في بداية التكرار التالي، فإنَّ الحالة الأولى لا تكون قد تسببت بخرق الخاصية 2. صحَّحت الحالةُ الأولى الخرقَ الوحيد للخاصية 4 الذي كان موجودًا في بداية هذا التكرار. ثمَّ جعلت z' حمراء وتركت $z'.p$ كما هي. فإذا كانت $z'.p$ سوداء فلا خرق في الخاصية 4. أما إذا كانت $z'.p$ حمراء فإنَّ تلوين z' بالأحمر يتسبب بخرق واحد في الخاصية 4 بين z' و $z'.p$.

الحالة الثانية: z هو ابنٌ أيمن وعمُّه y اسود

الحالة الثالثة: z هو ابنٌ أيسر وعمُّه y اسود

في الحالتين الثانية والثالثة عم z وهو y لونه أسود. نُفرِّق بين الحالتين بحسب كون z ابنًا أيمن أو أيسر لـ $z.p$. تؤلِّف الأسطر 10-11 الحالة الثانية، التي تظهر في الشكل 6.13 مع الحالة الثالثة. في الحالة الثانية، العقدة z هي ابن أيمن لأبيه. نستخدم فورًا دورانًا إلى اليسار لتحويل الوضع إلى الحالة الثالثة (الأسطر 12-14)، التي تكون فيها العقدة z ابنًا أيسر. ولما كان كل من z و $z.p$ حمراوان، فإنَّ الدوران لا يؤثِّر على الارتفاع الأسود للعقد ولا على الخاصية 5. وسواءً دخلنا الحالة الثالثة مباشرةً أو عبر الحالة الثانية، فإنَّ عم z يكون أسود،



الشكل 6.13 الحالتان الثانية والثالثة للإجراء RB-INSERT-FIXUP. كما في الحالة الأولى، تُخرق الخاصية 4 إما في الحالة الثانية وإما في الثالثة لأن z وأباها $z.p$ حمراوان. لكل من الأشجار الفرعية α، β، γ و δ جذرٌ أسود (α و β و γ من الخاصية 4، و δ لأننا بغير ذلك نكون في الحالة الأولى)، ولكل منها الارتفاع الأسود نفسه. تتحول الحالة الثانية إلى الحالة الثالثة بدوران إلى اليسار يحافظ على الخاصية 5: جميع المسارات النازلة من عقدة إلى ورقة لها العدد نفسه من العقد السوداء. تسبِّب الحالةُ الثالثةُ بعضَ التغيير في الألوان ودورانًا إلى اليمين يحافظ أيضًا على الخاصية 5. ثم تنتهي حلقة **while** لأن الخاصية 4 محققة: لم يعد ثمة عقدتان حمراوان في سطر.

لأننا بغير ذلك نكون قد نفذنا الحالة الأولى. إضافةً إلى ذلك، فإن العقدة $z.p.p$ موجودة، لأننا بيّنّا أن هذه العقدة كانت موجودة عند تنفيذ السطرين 2 و 3، وبعد نقل z إلى المستوى الأعلى في السطر 10 ثم نقله إلى المستوى الأدنى في السطر 11، لم تتغير هوية $z.p.p$. وفي الحالة الثالثة، ننفّذ بعضَ التغيير في الألوان ودورانًا إلى اليمين يحافظ على الخاصية 5، وبذلك نكون قد انتهينا، لأنه لم يعد لدينا عقدتان حمراوان في أي سطر. هذا وإن الحلقة **while** لا تتكرر مرةً أخرى لأن $z.p$ أصبحت سوداء الآن.

نبيِّن الآن أن الحالتين الثانية والثالثة تحافظان على لامتغير الحلقة. (كما بيّنّا للتو، فإن $z.p$ ستكون سوداء بعد الاختبار التالي في السطر 1، ولن يُنفَّذ جسم الحلقة ثانيةً.)

أ. تجعل الحالة الثانية z تؤشر على $z.p$ ذات اللون الأحمر. لا تجري تغيُّرات أخرى على z أو على لونه في الحالتين الثانية والثالثة.

ب. تجعل الحالة الثالثة العقدة $z.p$ سوداء، فإذا كانت هي الجذر في بداية التكرار التالي، كان لونها أسود.

ت. كما في الحالة الأولى، تحافظ الحالتان الثانية والثالثة على الخصائص 1 و 3 و 5.

ولمّا لم تكن العقدة z هي الجذر في الحالتين الثانية والثالثة، فإننا نعلم أنه ليس هناك خرق للخاصية 2. فالحالتين الثانية والثالثة لا تسببان خرقًا للخاصية 2، لأن العقدة الوحيدة التي أصبحت حمراء تصبح ابنًا لعقدة سوداء نتيجة للدوران في الحالة الثالثة.

تُصحِّح الحالتان الثانية والثالثة الخرقَ الوحيد للخاصية 4، ولا تُدخلان خرقًا آخر.

بعد أن أظهرنا أن كل تكرار للحلقة يحافظ على اللامتغير، نكون قد برهنّا أنَّ الإجراء RB-INSERT-FIXUP يستعيد الخصائص الحمراء–السوداء استعادةً صحيحة.

التحليل

ما هو زمن تنفيذ الإجراء RB-INSERT؟ لما كان ارتفاع الشجرة الحمراء–السوداء المؤلفة من n عقدة هو $O(\lg n)$، فإنَّ الأسطر 1-16 من RB-INSERT تستغرق زمنًا $O(\lg n)$. في الإجراء RB-INSERT-FIXUP، تتكرر حلقة **while** فقط إذا نُفِّذت الحالة الأولى، ثم ينتقل المؤشر z مستويَيْن إلى الأعلى في الشجرة. لذلك فإنَّ العدد الكلي لمرات تنفيذ الحلقة **while** يمكن أن يكون $O(\lg n)$. وهكذا فإنَّ RB-INSERT يستغرق زمنًا إجماليًّا $O(\lg n)$. وهو مع ذلك لا يمكن أن يقوم بأكثر من دورانين لأن حلقة **while** تنتهي إذا نُفِّذت الحالة الثانية أو الثالثة.

تمارين

1-3.13

في السطر 16 من الإجراء RB-INSERT، نحدِّد لونَ العقدة المدرجة حديثًا بالأحمر. لاحظ أننا لو اخترنا

وضعها بالأسود، فإنَّ الخاصية 4 من خصائص الشجرة الحمراء–السوداء لن تُخرَق. لماذا لم نَختَرْ تلوين z بالأسود؟

2-3.13

أَظْهِر الأشجارَ الحمراء–السوداء الناتجة عن إدراج متتابع للمفاتيح 41 و 38 و 31 و 12 و 19 و 8 في شجرة حمراء–سوداء فارغة في البداية.

3-3.13

افترض أنَّ الارتفاع الأسود لكلٍّ من الأشجار الفرعية α و β و γ و δ و ε في الشكلين 5.13 و 6.13 هو k. علِّم كل عقدة في الشكلَيْن بارتفاعها الأسود للتحقُّق من أن التحويل المشار إليه يحافظ على الخاصية 5.

4-3.13

يشعر أحد المدرسين بالقلق من إمكان وضع اللون RED في $T.nil.color$ في الإجراء RB-INSERT-FIXUP، في هذه الحالة لا يسبِّب الاختبارُ في السطر 1 إنهاء الحلقة عندما تكون z هي الجذر. بيِّن أنَّ قلقَ المدرس لا أساس له بإثبات أنَّ RB-INSERT-FIXUP لا يضع RED في $T.nil.color$ أبدًا.

5-3.13

لنأخذ شجرة حمراء–سوداء تكوَّنت من إدراج n عقدة باستخدام RB-INSERT. بيِّن أنَّه إذا كان $n > 1$ كانت في الشجرة عقدة واحدة حمراء على الأقل.

6-3.13

اقترح طريقة لتنجيز RB-INSERT بفعالية إذا لم يتضمن تمثيل الأشجار الحمراء–السوداء تخزين مؤشرات إلى الأب.

4.13 الحذف

كما في العمليات الأساسية الأخرى على شجرة حمراء–سوداء ذات n عقدة، يستغرق الحذف زمنًا $O(\lg n)$، إلا أنَّ حذف عقدة من شجرة حمراء–سوداء أكثر تعقيدًا بقليل من إدراج عقدة.

يعتمد إجراء حذف عقدة من شجرة حمراء–سوداء على الإجراء TREE-DELETE (المقطع 3.12). نحتاج أولاً إلى مواءمة الإجراء الفرعي TRANSPLANT الذي يستدعيه TREE-DELETE بحيث يُطبَّق على شجرةٍ حمراء–سوداء:

```
RB-TRANSPLANT(T, u, v)
1  if u.p == T.nil
2      T.root = v
3  elseif u == u.p.left
```

```
4        u.p.left = v
5   else u.p.right = v
6   v.p = u.p
```

يختلف الإجراء RB-TRANSPLANT عن TRANSPLANT من ناحيتين. الأولى أنَّ السطر 1 يستخدم الحارس $T.nil$ بدلاً من NIL، والثانية أن الإسناد إلى $v.p$ يحدث في السطر 6 دون شرط: يمكننا الإسناد إلى $v.p$ وإن كان v يشير إلى الحارس. في الحقيقة سنستفيد من إمكان الإسناد إلى $v.p$ عندما يكون $v = T.nil$.

إن الإجراءَ RB-DELETE يشبه الإجراءَ TREE-DELETE، ولكن بزيادة أسطر من شبه الرماز. تقتفي بعض الأسطر الإضافية أثر عقدة y قد تتسبب في خرق الخصائص الحمراء–السوداء. وعندما نريد حذف عقدة z التي لها أقل من ابنين، فإنها تُحذف من الشجرة ونريد أن تصبح y هي z. أما عندما يكون لـ z ابنان، فيجب عندها أن تكون y هي العقدة التالية لـ z، وأن تحتل y موضع z في الشجرة. كذلك نسجِّل لون y قبل حذفها من الشجرة أو انتقالها ضمنها، ونتتبَّع العقدة x التي تنتقل إلى موضع y الأصلي في الشجرة، لأن العقدة x قد تسبب أيضًا خروقًا في الخصائص الحمراء–السوداء. بعد حذف العقدة z، يستدعي الإجراء RB-DELETE إجراءً مساعدًا RB-DELETE-FIXUP يغيِّر الألوانَ ويقوم بدوراناتٍ لاستعادة الخصائص الحمراء–السوداء.

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MIINMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

ومع أنَّ RB-DELETE يتضمن عددًا من أسطر شبه الرماز يقارب ضعف عدد أسطر TREE-DELETE، فإنَّ الإجراءين لهما البنية الأساسية نفسها. يمكنك إيجاد كل سطر من TREE-DELETE ضمن RB-DELETE (بتغيير NIL إلى $T.nil$ والاستعاضة عن كل استدعاءات TRANSPLANT باستدعاءات RB-TRANSPLANT)، منفَّذة ضمن الشروط نفسها.

وفيما يلي بقية الاختلافات بين الإجراءين:

- نحافظ على العقدة y على أنها العقدة التي يجري حذفها من الشجرة أو نقلها ضمنها. يجعل السطر 1 العقدةَ y تشير إلى العقدة z عندما يكون لـ z أقل من ابنين، ومن ثم فهي تُحذف. أما عندما يكون لـ z ابنان، فإننا نجد في السطر 9 أن y تشير إلى العقدة التي تلي z كما في TREE-DELETE تمامًا، وستحتل y موضع z في الشجرة.
- لما كان من الممكن أن يتغيَّر لون y، فإن المتحول *y-original-color* يخزن لونها قبل حدوث أي تغيير. يقوم السطران 2 و 10 بتحديد قيمة لهذا المتحول مباشرةً بعد الإسناد إلى y . عندما يكون لـ z ابنان، فإن $y \neq z$، وتنتقل العقدة y إلى الموضع الأصلي للعقدة z في الشجرة الحمراء–السوداء؛ يعطي السطر 20 للعقدة y لون z نفسه. ونحتاج إلى حفظ لون y الأصلي لاختباره في نهاية RB-DELETE؛ فإذا كان أسود فإنَّ حذف y أو نقلها قد يسبب خروقات في الخصائص الحمراء–السوداء.
- كما ذكرنا آنفًا، نتتبع العقدة x التي تنتقل إلى موضع y الأصلي. أما الإسنادات في الأسطر 4 و 7 و 11 فتجعل x تشير إلى ابن y الوحيد أو إلى الحارس $T.nil$ إذا لم يكن لـ y أولاد (تذكَّر من المقطع 3.12 أنَّ y ليس لها ابن أيسر).
- لما كانت العقدة x تنتقل إلى موضع العقدة y الأصلي، فإنَّ الواصفة $x.p$ مهيأة دومًا لتشير إلى الموضع الأصلي لأبي y في الشجرة، وإن كانت العقدة x هي في الحقيقة الحارس $T.nil$. وفيما عدا الحالة التي يكون فيها z هو الأب الأصلي للعقدة y (وهذا يحدث فقط عندما يكون للعقدة z ابنان وتكون العقدة التالية لها y هي ابنها الأيمن)، يحدث الإسناد إلى $x.p$ في السطر 6 من RB-TRANSPLANT. (لاحظ أنَّه عند استدعاء RB-TRANSPLANT في الأسطر 5 أو 8 أو 14 فإنَّ الموسط الثالث الذي يجري تمريره يماثل x.)

  مع ذلك، عندما يكون الأب الأصلي للعقدة y هو z، فإننا لا نريد أن يشير $x.p$ إلى الأب الأصلي للعقدة y، لأننا بصدد حذف تلك العقدة من الشجرة. ولما كانت العقدة y ستنتقل إلى الأعلى لتأخذ موضع z في الشجرة، فإنَّ وضع y في $x.p$ في السطر 13 سيجعل $x.p$ يشير إلى الموضع الأصلي لأبي العقدة y وإن كان $x = T.nil$.
- أخيرًا، إذا كانت العقدة y سوداء، فقد نكون أحدثنا خرقًا أو أكثر في الخصائص الحمراء–السوداء،

ولذلك نستدعي RB-DELETE-FIXUP في السطر 22 لاستعادة الخصائص الحمراء–السوداء. فإذا كانت y حمراء، بقيت الخصائص الحمراء–السوداء محقَّقةً عند حذف y أو نقلها، وذلك للأسباب التالية:

1. لم تتغيَّر أية ارتفاعات سوداء في الشجرة.
2. لم يجرِ وضع عقد حمراء متجاورة. ولما كانت y تأخذ مكان z في الشجرة وكذلك لون z، فلا يمكن أن يصبح لدينا عقدتان حمراوان متجاورتان في موضع y الجديد في الشجرة. إضافةً إلى ذلك، إذا لم تكن y الابن الأيمن للعقدة z، فإنَّ الابن الأيمن الأصلي لها x سيحل محلها في الشجرة. فإذا كانت y حمراء، لزم أن يكون x أسود، وبذلك لا يمكن أن يسبِّب تبديل x بـ y في أن تصبح عقدتان حمراوان متجاورتين.
3. لما لم يكن بإمكان y أن تكون جذرًا إذا كانت حمراء، فإنَّ الجذر يبقى أسود.

إذا كانت العقدة y سوداء، فيمكن أن تظهر ثلاث مشكلات تُحَلُّ باستدعاء RB-DELETE-FIXUP. أولاً، إذا كانت y هي الجذر وأصبح ابنٌ أحمر لها هو الجذر الجديد، فنكون قد خرقنا الخاصية 2. ثانيًا، إذا كان كلٌّ من x و $x.p$ حمراوين، فنكون قد خرقنا الخاصية 4. ثالثًا، إنَّ نقل y ضمن الشجرة يسبِّب نقص عقدةٍ سوداء من كل مسارٍ بسيط كان يتضمن y. وبذلك تُخرق الخاصية 5 من كل أسلاف y في الشجرة. نصحِّح خرق الخاصية 5 بالقول إنَّ العقدة x التي تشغل الآن الموضع الأصلي للعقدة y لديها عقدةٌ سوداء "إضافية". أي إننا إذا أضفنا 1 إلى عدد العقد السوداء في أي مسارٍ بسيط يتضمن x، فإنَّ الخاصية 5 تتحقق بحسب هذا التفسير. وعندما نحذف أو ننقل العقدة السوداء y، "ندفع" بسوادها في العقدة x. فالمشكلة الآن هي أنَّ العقدة x لم تعد حمراء ولا سوداء، وذلك يخرق الخاصية 1. وبدلاً من ذلك، إما أن تكون العقدة x "سوداء مضاعفة" وإما "حمراء وسوداء" وتسهم في عدد العقد السوداء في المسارات البسيطة التي تحتوي x بمقدار 2 أو 1 بالترتيب. وسيبقى واصف اللون $color$ في العقدة x إما RED (إذا كانت x حمراء وسوداء) وإما BLACK (إذا كانت x سوداء مضاعفة). وبتعبير آخر فإنَّ الأسود الإضافي في عقدة ما يؤثِّر في تأشير x عليها، وليس في واصف اللون $color$.

يمكننا الآن معاينة الإجراء RB-INSERT-FIXUP وتفحُّص كيفية إعادته الخصائصَ الحمراء–السوداء إلى شجرة البحث.

```
RB-DELETE-FIXUP(T, x)
1  while x ≠ T.root and x.color == BLACK
2      if x == x.p.left
3          ω = x.p.right
4          if ω.color == RED
5              ω.color = BLACK                          // Case 1
6              x.p.color = RED                          // Case 1
```

```
7                  LEFT-ROTATE(T, x.p)                                    // Case 1
8                  ω = x.p.right                                          // Case 1
9             if ω.left.color == BLACK and ω.right.color == BLACK
10                 ω.color = RED                                          // Case 2
11                 x = x.p                                                // Case 2
12            else if ω.right.color == BLACK
13                     ω.left.color = BLACK                               // Case 3
14                     ω.color = RED                                      // Case 3
15                     RIGHT-ROTATE(T, ω)                                 // Case 3
16                     ω = x.p.right                                      // Case 3
17                 ω.color = x.p.color                                    // Case 4
18                 x.p.color = BLACK                                      // Case 4
19                 ω.right.color = BLACK                                  // Case 4
20                 LEFT-ROTATE(T, x.p)                                    // Case 4
21                 x = T.root                                             // Case 4
22        else (same as then clause with "right" and "left" exchanged)
23   x.color = BLACK
```

يستعيد الإجراء RB-DELETE-FIXUP الخواص 1 و 2 و 4. يُطلب إليك في التمرينين 13.4-1 و 13.4-2 أن تثبت أنَّ الإجراء يستعيد الخاصيتين 2 و 4، لذلك فإنَّنا سنركِّز في ما تبقى من هذا المقطع على الخاصية 1. ويتمثَّل الغرض من حلقة **while** في الأسطر 1-22 في نقل السواد الإضافي إلى أعلى الشجرة إلى أن:

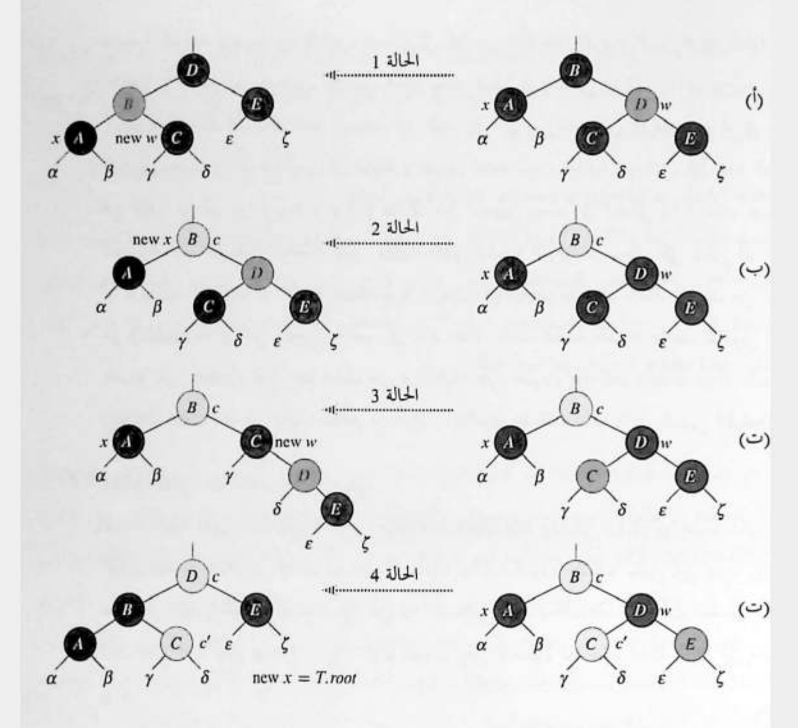
1. يشير x إلى عقدة حمراء-سوداء، وفي هذه الحالة نلوّن x فقط بالأسود في السطر 23.
2. يشير x إلى الجذر، وفي هذه الحالة "نحذف" السواد الإضافي فقط.
3. نخرج من الحلقة بعد إجراء دورانات وعمليات إعادة تلوين مناسبة.

وضمن الحلقة **while**، تشير x دومًا إلى عقدة سوداء مضاعفة لا تنتمي إلى الجذر. نُحدِّد في السطر 2 إذا كان x هو ابنًا أيسرَ أو أيمنَ لأبيه $x.p$. (أعطينا الرماز للحالة التي يكون فيها x ابنًا أيسر؛ أما الحالة التي يكون فيها x ابنًا أيمن —السطر 22— فهي حالة مناظرة.) نحافظ على مؤشر ω على العقدة المجاورة لـ x. لما كانت x ذات سواد مضاعف، فإنَّ العقدة ω لا يمكن أن تكون $T.nil$؛ وإلا فإنَّ عدد العقد السوداء على المسار البسيط من $x.p$ إلى الورقة ω (ذات السواد الوحيد) سيكون أصغر من عددها في المسار البسيط من $x.p$ إلى x.

تَظهر الحالات الأربع[2] الموجودة في الرماز في الشكل 13.7. وقبل دراسة كل حالة بالتفصيل، لنلق نظرة عامة على كيفية التحقُّق من حفاظ التحويل في كل حالة على الخاصية 5. الفكرة الرئيسية هي أنَّ التحويل المطبَّق يحافظ على عدد العقد السوداء (ومنها السواد الإضافي الخاص بـ x) في كل حالة انطلاقًا من جذر

---

[2] كما في RB-INSERT-FIXUP، ليست الحالات في RB-DELETE-FIXUP حالات إقصاء متبادل.



الشكل 7.13 حالات الحلقة **while** في الإجراء RB-DELETE-FIXUP. العقد المظللة تكون واصفة اللون *color* فيها BLACK، أما العقد المظللة تظليلاً كثيفًا فواصفة اللون *color* فيها RED، أما العقد المظللة تظليلاً فاتحًا فواصفة اللون فيها ممثلة بـ c أو c'، الذي يمكن أن يكون RED أو BLACK. تمثّل الأحرف $\alpha, \beta, \ldots, \zeta$ أشجارًا فرعية اعتباطية. كل حالة تقوم بتحويل التشكيلة الموجودة إلى اليسار إلى التشكيلة الموجودة إلى اليمين، بتغيير بعض الألوان و/أو إجراء دوران. أي عقدة تشير إليها x يكون فيها سوادٌ إضافي، وتكون إما سوداء مضاعفة وإما حمراء–سوداء. الحالة الوحيدة التي تسبب تكرار الحلقة هي الحالة الثانية case 2. (أ) يجري تحويل الحالة الأولى إلى الحالات 2 أو 3 أو 4 بتبديل ألوان العقدتين B و D وإجراء دوران إلى اليسار. (ب) في الحالة الثانية يجري نقل السواد الإضافي المتمثل بالمؤشر x إلى أعلى الشجرة بتلوين العقدة D بالأحمر وجعل x تؤشر على العقدة B. وإذا دخلنا الحالة الثانية عبر الحالة الأولى، فإنَّ الحلقة **while** تنتهي لأن العقدة الجديدة x هي حمراء–سوداء، ومن ثَمَّ فإنَّ القيمة c لواصفة اللون فيها هي RED. (ت) تُحوَّل الحالة الثالثة إلى الحالة الرابعة بتبديل ألوان العقدتين C و D وإجراء دوران يميني. (ث) تحذف الحالة الرابعة السواد الإضافي المتمثلة بالمؤشر x بتغيير بعض الألوان وإجراء دوران إلى اليسار (دون خرق الخصائص الحمراء–السوداء)، وتنتهي الحلقة.

الشجرة الفرعية المرسومة (وباعتباره متضمنًا) إلى كلٍّ من الأشجار الفرعية $\alpha, \beta, \ldots, \zeta$. لذلك إذا كانت الخاصية 5 محققة قبل التحويل، فستظل محقَّقةً بعده. فمثلاً في الشكل 7.13(أ) الذي يوضّح الحالة الأولى case 1، نلاحظ أنَّ عدد العقد السوداء من الجذر إلى أي من الشجرتين الفرعيتين α أو β هو 3، قبل التحويل وبعده. (تذكَّر مرةً أخرى أنَّ العقدة x تضيف عقدة سوداء إضافية.) وبالمثل، فإنَّ عدد العقد السوداء من الجذر إلى أيٍ من γ و δ و ε و ζ هو 2، قبل التحويل وبعده. في الشكل 7.13(ب)، يجب أن يأخذ العد بالحسبان القيمة c لواصفة اللون *color* لجذر الشجرة الفرعية المرسومة، التي يمكن أن تكون RED أو BLACK. إذا عرَّفنا count(RED) = 0 و count(BALCK) = 1، فإنَّ عدد العقد السوداء من الجذر إلى α هو 2 + count(c)، قبل التحويل وبعده. في هذه الحالة، تأخذ العقدة الجديدة x بعد التحويل القيمة c في واصفة اللون *color*، لكن هذه العقدة هي في الحقيقة حمراء–سوداء (إذا كان c = RED) أو سوداء مضاعفة (إذا كان c = BLACK). يمكن التحقُّق من الحالات الأخرى بطريقة مشابهة (انظر التمرين 13.4-5).

الحالة الأولى: ω المجاورة لـ x حمراء

تحدث الحالة الأولى (الأسطر 5-8 من RB-DELETE-FIXUP والشكل 7.13(أ)) عندما تكون العقدة ω المجاورة للعقدة x حمراء. وإذ إن ω يجب أن يكون لها أبناء سود، فيمكننا تبديل لوني ω و $x.p$ ثم إجراء دوران إلى اليسار حول $x.p$ دون خرق أيٍ من الخصائص الحمراء–السوداء. أصبحت العقدة الجديدة المجاورة لـ x، وهي أحد أبناء ω قبل الدوران، الآن سوداء، ومن ثَمَّ نكون قد حوَّلنا الحالة الأولى إلى الحالة 2 أو 3 أو 4.

تحدث الحالات 2 و 3 و 4 عندما تكون العقدة ω سوداء؛ وتتمايز بألوان أبناء ω.

الحالة الثانية: ω المجاورة لـ x سوداء، وابناها أسودان

في الحالة الثانية (الأسطر 10-11 من RB-DELETE-FIXUP والشكل 7.13(ب)) يكون ابنا ω أسودان. ولما كانت ω أيضًا سوداء، فنأخذ سوادًا واحدًا من x ومن ω، لتصبح x بسواد واحد وتصبح ω حمراء. للتعويض عن حذف سواد واحد من x ومن ω، نريد أن نضيف سوادًا إضافيًا إلى $x.p$، الذي كان في الأصل أحمر أو أسود. نقوم بذلك بتكرار الحلقة **while** مع اعتبار $x.p$ هي العقدة الجديدة x. لاحظ أننا إذا دخلنا الحالة الثانية عبر الحالة الأولى، فإنَّ العقدة الجديدة x هي حمراء–سوداء، لأن العقدة الأصلية $x.p$ كانت حمراء. إذن فالقيمة c لواصفة اللون *color* في العقدة الجديدة x هي RED، وتنتهي الحلقة عندما تختبر شرط التكرار. ثم نلوِّن العقدةَ الجديدة x بلون أسود (وحيد) في السطر 23.

الحالة الثالثة: ω المجاورة لـ x سوداء، وابنها الأيسر أحمر وابنها الأيمن أسود

تحدث الحالة الثالثة (الأسطر 13-16 والشكل 7.13(ت)) عندما تكون ω سوداء، وابنها الأيسر أحمر وابنها

الأيمن أسود. يمكننا تبديل لوني ω وابنها الأيسر $\omega.left$ ثم إجراء دوران إلى اليمين حول ω دون خرق أيٍّ من الخصائص الحمراء-السوداء. العقدة الجديدة ω المجاورة لـ x هي الآن عقدة سوداء لها ابن أيمن أحمر، ومن ثَمَّ نكون قد حوَّلنا الحالة 3 إلى الحالة 4.

الحالة الرابعة: ω المجاورة لـ x سوداء، وابنها الأيمن أحمر

تحدث الحالة الرابعة (الأسطر 17-21 والشكل 7.13(ث)) عندما تكون العقدة ω المجاورة لـ x سوداء وابنها الأيمن أحمر. بإجراء بعض التغييرات في الألوان وتنفيذ دوران إلى اليسار حول $x.p$، يمكننا حذف الأسود الإضافي في x لتصبح وحيدة السواد دون خرق أيٍّ من الخصائص الحمراء-السوداء. وبافتراض أن x هي الجذر تنتهي حلقة **while** عند اختبارها شرطَ التكرار.

التحليل

ما هو زمن تنفيذ RB-DELETE؟ لمّا كان ارتفاع الشجرة الحمراء-السوداء المؤلفة من n عقدة هو $O(\lg n)$، فإنَّ التكلفة الإجمالية للإجراء دون استدعاء RB-DELETE-FIXUP تستغرق زمنًا $O(\lg n)$. مع الإجراء RB-DELETE-FIXUP، تنتهي الحالات 1 و 3 و 4 بعد أداء عدد ثابت من تغييرات اللون وثلاثة دورانات على الأكثر. والحالة 2 هي الحالة الوحيدة التي يمكن أن تتكرر فيها الحلقة **while**، ومن ثم ينتقل المؤشر x إلى أعلى الشجرة عددًا من المرات لا يتجاوز $O(\lg n)$ مرة، وبدون دورانات. لذلك، فإنَّ الإجراء RB-DELETE-FIXUP يستغرق زمنًا $O(\lg n)$، ويؤدي ثلاثة دورانات على الأكثر، ويكون الزمن الإجمالي للإجراء RB-DELETE إذن $O(\lg n)$ أيضًا.

تمارين

1-4.13

برهن أنَّه بعد تنفيذ RB-DELETE-FIXUP، يجب أن يكون جذر الشجرة أسود.

2-4.13

برهن أنه إذا كان كل من x و $x.p$ حمراوان في RB-DELETE، فإنَّ الخاصية 4 تُسترَد باستدعاء RB-DELETE-FIXUP(T, x).

3-4.13

أوجدتَ في التمرين 2-3.13 الشجرةَ الحمراء-السوداء التي تنتج عن إدراج متتابع للمفاتيح التالية 41, 38, 31, 12, 19, 8 في شجرة فارغة في البداية. بيِّن الآن الأشجار الحمراء-السوداء التي تنتج عن الحذف المتتابع للمفاتيح بحسب الترتيب 8, 12, 19, 31, 38, 41.

4-4.13

في أية أسطر من رماز RB-DELETE-FIXUP يمكننا تفحص أو تعديل الحارس $T.nil$؟

5-4.13

في كل من حالات الشكل 7.13، حدِّد عدد العقد السوداء انطلاقًا من جذر الشجرة الفرعية المرسومة إلى كلٍّ من الأشجار الفرعية $\alpha, \beta, \ldots, \zeta$ وتأكد أن عددها يظل نفسه بعد التحويل. وإذا كانت واصفة اللون ***color*** لعقدة ما هي c أو c'، فاستخدم التدوين $\text{count}(c)$ أو $\text{count}(c')$ رمزيًّا في إجابتك.

6-4.13

يعتقد الأستاذان Skelton و Baron بأنَّ لون العقدة $x.p$ يمكن أن لا يكون أسود في بداية الحالة الأولى من RB-DELETE-FIXUP. فإذا كان الأستاذان محقَّيْن، تكون الأسطر 5-6 خاطئة. أثبت أن $x.p$ يجب أن يكون أسود في بداية الحالة الأولى، بحيث يدرك الأستاذان أنه ليس ثمة ما يقلقا بشأنه.

7-4.13

افترض أنَّ عقدةً x أُدرجت في شجرة حمراء–سوداء باستخدام RB-INSERT، ثم حُذفت مباشرةً باستخدام RB-DELETE. هل الشجرة الحمراء–السوداء الناتجة هي نفسها الشجرة الحمراء–السوداء الأولى؟ علِّل جوابك.

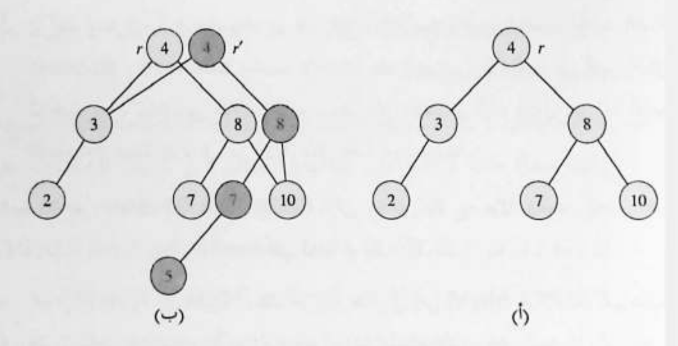
مسائل

1-13 مجموعات ديناميكية دائمة

في سياق تتبع خوارزميةٍ ما، نجد أحيانًا أننا نحتاج إلى الاحتفاظ بالنسخ السابقة لمجموعة ديناميكية dynamic set عند تعديلها. تُسمى هذه المجموعة ***دائمة persistent***. تعتمد إحدى طرق تنجيز مجموعة دائمة على نَسْخ المجموعة كاملةً كلما جرى تعديلها، لكن هذا النهج يمكن أن يُبطئ البرنامج وأن يستهلك حجمًا كبيرًا. على أنه يمكننا أحيانًا أن نفعل شيئًا أفضل بكثير من هذا النهج.

افترض S مجموعة دائمة مع العمليات INSERT, DELETE, SEARCH التي ننجزها باستخدام أشجار بحث ثنائية كما في الشكل 8.13(أ). نحتفظ بجذر منفصل لكل نسخة من المجموعة. ولإدراج المفتاح 5 في المجموعة، ننشئ عقدة جديدة مفتاحها 5. تصبح هذه العقدة الابن الأيسر لعقدة جديدة مفتاحها 7، لأننا لا نستطيع تغيير العقدة الموجودة سابقًا ذات المفتاح 7. وبالمثل، فإنَّ العقدة الجديدة ذات المفتاح 7 تصبح الابن الأيسر لعقدة جديدة مفتاحها 8 وابنها الأيمن هو العقدة الموجودة ذات المفتاح 10. تصبح العقدة الجديدة ذات المفتاح 8 بدورها الابن الأيمن لجذر جديد r' ذي المفتاح 4 وله ابن أيسر هو عقدة موجودة مفتاحها 3. وهكذا، فنحن ننسخ فقط جزءًا من الشجرة ونشترك ببعض العقد مع الشجرة الأصلية، كما يظهر في الشكل 8.13(ب).

لنفترض أنَّ لكل عقدة في الشجرة الواصفات key و $left$ و $righ$، ولكن ليس لها أب. (انظر أيضًا التمرين 6-3.13).



الشكل 8.13 (أ) شجرة بحث ثنائية مفاتيحها 10 ,8 ,7 ,4 ,3 ,2. (**ب**) شجرة البحث الثنائية الدائمة الناتجة عن إدراج المفتاح 5. تتألف أحدث نسخةٍ من مجموعة من العقد التي يمكن الوصول إليها من الجذر r'، وتتألف النسخة السابقة من العقد التي يمكن الوصول إليها من r. أضيفت العقد المظللة تظليلاً كثيفًا عند إدراج المفتاح 5.

أ. حدِّد العقد التي نحتاج إلى تغييرها لإدراج مفتاح k، أو حذف عقدة y في شجرة بحث ثنائية دائمة عامة.

ب. إذا كان لديك شجرة دائمة T ومفتاح k للإدراج، فاكتب إجراءً PERSISTENT-TREE-INSERT يعيد شجرةً جديدة دائمة T' ناتجة عن إدراج k في T.

ت. إذا كان ارتفاع شجرة البحث الثنائية الدائمة T هو h، فما هي متطلبات الزمن والحجم لتنجيز PERSISTENT-TREE-INSERT؟ (تتناسب متطلبات الحجم مع عدد العقد الجديدة المخصصة.)

ث. افترض أنَّنا ضمنَّا واصفة parent في كل عقدة. في هذه الحالة سيحتاج PERSISTENT-TREE-INSERT إلى القيام بعمليات نسخ إضافية. أثبت أنَّ PERSISTENT-TREE-INSERT يستغرق زمنًا وحجمًا $\Omega(n)$، حيث n هو عدد العقد في الشجرة.

ج. أظهر كيفية استخدام الأشجار الحمراء-السوداء لضمان بقاء الزمن والحجم اللازمين للتنفيذ في أسوأ الحالات $O(\lg n)$ لكل عملية إدراج أو حذف.

2-13 عملية الضم على الأشجار الحمراء-السوداء

تأخذ عملية *الضم join* مجموعتين ديناميكيتين S_1 و S_2 وعنصرًا x بحيث يكون لكل $x_1 \in S_1$ و $x_2 \in S_2$، لدينا $x_1.key \le x.key \le x_2.key$. تعيد هذه العملية مجموعة $S = S_1 \cup \{x\} \cup S_2$. نتحرى في هذه المسألة كيفية تنجيز عملية الضم في الأشجار الحمراء-السوداء.

أ. إذا كان لدينا شجرة حمراء-سوداء T، نخزن ارتفاعها الأسود باعتباره الواصف الجديد $T.bh$. بيِّن أنَّ RB-INSERT وRB-DELETE يمكنهما الاحتفاظ بهذا الواصف دون الحاجة إلى تخزين إضافي في عقد الشجرة، ودون زيادة زمن التنفيذ المقارب. برهنْ أنه يمكننا، في أثناء النزول عبر T، تحديد الارتفاع الأسود لكل عقدة نزورها في زمن $O(1)$ لكل عقدة جرت زيارتها.

نرغب بتنجيز عملية $\text{RB-JOIN}(T_1, x, T_2)$ التي تدمِّر T_1 و T_2 وتعيد شجرة حمراء-سوداء $T = T_1 \cup \{x\} \cup T_2$. ليكن n العدد الكلي للعقد في T_1 و T_2.

ب. افترض أنَّ $T_1.bh \geq T_2.bh$. صف خوارزمية تعمل في زمن $O(\lg n)$ بإمكانها أن تجد عقدة سوداء y في T_1 يكون مفتاحها هو الأكبر بين العقد التي لها الارتفاع الأسود $T_2.bh$.

ت. لتكن T_y الشجرة الفرعية ذات الجذر y. صف كيف يمكن أن تَحل $T_y \cup \{x\} \cup T_2$ محل T_y في زمن $O(1)$ دون تدمير خصائص شجرة البحث الثنائية.

ث. ما اللون الذي يمكن أن نلوّن به x بحيث نحافظ على الخصائص الحمراء-السوداء 1 و 3 و 5؟ صف كيف يمكن فرض الخصائص 2 و 4 في زمن $O(\lg n)$.

ج. برهن أن الفرض في الجزء (ب) لا يضيع العمومية. صِف الحالة المناظرة التي تَظهر عندما يكون $T_1.bh \leq T_2.bh$.

ح. برهن أنَّ زمن تنفيذ RB-JOIN هو $O(\lg n)$.

13-3 أشجار *AVL*

شجرة AVL هي شجرة بحث ثنائية ***متوازنة الارتفاع height balanced***: ففي كل عقدة x، تختلف ارتفاعات الأشجار الفرعية اليمنى واليسرى بمقدار 1 على الأكثر. ولتنجيز شجرة AVL، نحتفظ بواصفٍ إضافي في كل عقدة: $x.h$ هو ارتفاع العقدة x. ونفترض، كما في أية شجرة بحث ثنائية أخرى، أنَّ $T.root$ يشير إلى عقدة الجذر.

أ. برهن أنَّ ارتفاع شجرة AVL ذات n عقدة هو $O(\lg n)$. (*تلميح*: برهن أنَّه في شجرة AVL ارتفاعها h، هناك على الأقل F_h عقدة، حيث F_h هو رقم فيبوناتشي Fibonacci من الرتبة h.)

ب. لإدراج عقدة ضمن شجرة AVL، يجري أولاً وضع هذه العقدة في مكان مناسب ضمن ترتيب شجرة البحث الثنائية. بعد ذلك قد لا تبقى الشجرة متوازنة الارتفاع. وبالتحديد قد يختلف ارتفاعا الابنين الأيسر والأيمن لبعض العقد بمقدار 2. صف إجراءً $\text{BALANCE}(x)$ يأخذ شجرة فرعية جذرها x، وابناها الأيمن والأيسر متوازنا الارتفاع وتختلف ارتفاعاتهما بمقدار 2 على الأكثر، أي إنَّ

$|x.right.h - x.left.h| \leq 2$، ويعدِّل الشجرة الفرعية ذات الجذر x لتصبح متوازنة الارتفاع. (تلميح: استخدم الدورانات.)

ت. باستخدام الجزء (ب)، صف إجراءً عوديًا AVL-INSERT(x, z) يأخذ عقدة x ضمن شجرة AVL وعقدة منشأة حديثًا z (سبق ملء مفتاحها)، ويضيف z إلى الشجرة الفرعية ذات الجذر x، محافظًا على خاصية أن x هي جذر شجرة AVL. كما في الإجراء TREE-INSERT من المقطع 3.12 ، افترض أنَّ $z.key$ قد مُلئ سابقًا، وأنَّ $z.left =$ NIL و $z.right =$ NIL؛ وافترض أيضًا أنَّ $z.h = 0$. إذن لإدراج العقدة z في شجرة AVL وهي T، نستدعي AVL-INSERT$(T.root, z)$.

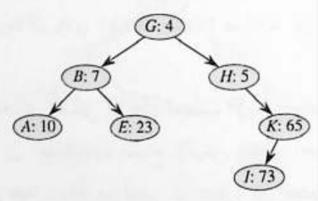
ث. بيِّن أنَّ تنفيذ عملية AVL-INSERT على شجرة AVL ذات n عقدة يستغرق زمنًا $O(\lg n)$، ويؤدي $O(1)$ دورانًا.

4-13 *الأشجار المكوَّمة*

إذا أدرجنا مجموعةً من n عنصرًا في شجرة بحث ثنائية، فقد تكون الشجرة الناتجة شديدة الاختلال في التوازن، وهذا يؤدي إلى أزمنة بحث طويلة. ومع ذلك، وكما رأينا في المقطع 4.12 فإنَّ أشجار البحث الثنائية المبنية عشوائيًّا تميل إلى أن تكون متوازنة. لذلك يمكن اعتماد استراتيجية تبني – وسطيًّا – شجرة متوازنة من مجموعة ثابتة من العناصر تقوم على إجراء تبديل عشوائي للعناصر ثم إدراجها في الشجرة وفق ذلك الترتيب.

ولكن ماذا يحدث لو لم تكن لدينا كل العناصر دفعةً واحدة؟ إذا كنا نستقبل العناصر واحدًا تلو الآخر، فهل يمكننا بناء شجرة بحث ثنائية عشوائيًّا بها؟

سندرس بنية معطيات تعطي جوابًا إيجابيًّا عن هذا السؤال. ***الشجرة المكوَّمة treap*** هي شجرة بحث ثنائية لها طريقة معدَّلة في ترتيب العقد. يُظهر الشكل 9.13 مثالاً عنها. وكالمعتاد، تملك كل عقدة x في الشجرة مفتاحًا قيمته $x.key$. إضافةً إلى ذلك، نسند الأفضلية $x.priority$ وهي رقم عشوائي نختاره اختيارًا مستقلاً لكل عقدة. نفترض أنَّ كل الأفضليات متمايزة، وأن كل المفاتيح متمايزة أيضًا. تكون عقد الشجرة المكوَّمة مرتبة بحيث تحقق المفاتيح خصائص أشجار البحث الثنائية، وتحقق الأفضلياتُ خاصيةَ ترتيب الكومة وفق الأصغر:



الشكل 9.13 شجرة مكوَّمة. كل عقدة x معلَّمة بـ $x.key : x.Priority$. فالجذر مثلاً مفتاحه G وأفضليته 4.

- إذا كان v ابنًا أيسر لـ u، فإنَّ $v.key < u.key$.
- إذا كان v ابنًا أيمن لـ u، فإنَّ $v.key > u.key$.
- إذا كان v ابنًا لـ u، فإنَّ $v.priority > u.priority$.

(سميت الشجرة بالشجرة المكوَّمة بسبب تركيب مجموعة الخصائص هذه؛ فهي تملك في آنٍ معًا ميزات شجرة البحث الثنائية والكومة.)

من المفيد النظر إلى الأشجار المكوَّمة على النحو الآتي. افترض أننا نُدرج عقدًا $x_1, x_2, \ldots, x_n$ مع مفاتيحها المرافقة ضمن شجرة مكوَّمة. فالشجرة المكوَّمة الناتجة هي الشجرة التي كانت ستتشكَّل فيما لو كانت العقد قد أُدرجت في شجرة بحث ثنائية عادية وفق ترتيب أفضلياتها (المختارة عشوائيًا). أي إنَّ $x_i.priority < x_j.priority$ تعني أننا أدرجنا x_i قبل x_j.

أ. لتكن لدينا مجموعة العقد $x_1, x_2, \ldots, x_n$ مع مفاتيحها المرافقة وأفضلياتها، وجميعها متمايزة. بيِّن أن الشجرة المكوَّمة المرافقة لهذه العقد وحيدة.

ب. برهن أنَّ الارتفاع المتوقَّع لشجرة مكوَّمة هو $\Theta(\lg n)$، وأنَّ الزمن المتوقَّع للبحث عن قيمة في الشجرة المكوَّمة هو $\Theta(\lg n)$.

لنرَ كيف يمكن إدراج عقدة جديدة في شجرة مكوَّمة موجودة. نبدأ بإسناد أفضلية عشوائية للعقدة الجديدة. ثم نستدعي خوارزمية الإدراج، التي نسميها TREAP-INSERT ونوضِّح كيفية تشغيلها في الشكل 10.13.

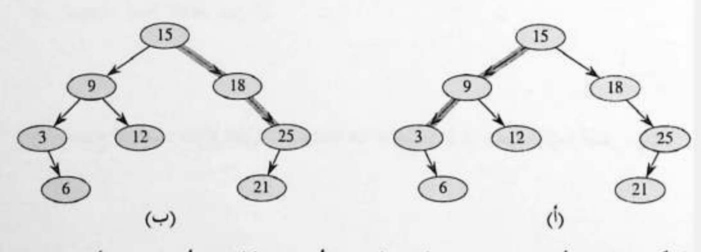
ت. اشرح كيفية عمل TREAP-INSERT، واكتب شبه الرماز اللازم. (*تلميح:* نفِّذ إجراء الإدراج المعتاد في شجرة البحث الثنائية، ثم أجرِ بعض الدورانات لاستعادة خاصية ترتيب الكومة وفق الأصغر.)

ث. بيِّن أنَّ زمن التنفيذ المتوقع لـ TREAP-INSERT هو $\Theta(\lg n)$.

ينفِّذ TREAP-INSERT بحثًا ثم متتالية من الدورانات. ومع أنَّ هاتين العمليتين لهما نفس الزمن المتوقع للتنفيذ، فإنَّ لهما عمليًا تكلفة مختلفة. فالبحث يقرأ معلومات من الشجرة المكوَّمة دون تعديلها، على حين يغيِّر الدوران مؤشرات الأب والابن ضمن الشجرة المكوَّمة. ويلاحظ أنه في معظم الحواسيب، تكون عمليات القراءة أسرع بكثير من عمليات الكتابة، لذا فنحن نريد أن يؤدي TREAP-INSERT دورانات أقل. سنبيِّن أنَّ العدد المتوقع للدورانات محدود بثابت ما.

لإجراء ذلك، نحتاج إلى بعض التعاريف الموضَّحة في الشكل 11.13. ***العصب الأيسر left spine*** في شجرة بحث ثنائية T هو المسار البسيط من الجذر إلى العقدة ذات المفتاح الأصغر. وبتعبير آخر، العصب الأيسر هو المسار البسيط المؤلف من وصلات يسرى فقط انطلاقًا من الجذر. وبالتناظر، فإن ***العصب الأيمن***

الشكل 10.13 تشغيل TREAP-INSERT. (أ) الشجرة المكوَّمة الأصلية قبل الإدراج. (**ب**) الشجرة المكوَّمة بعد إدراج عقدة مفتاحها C وأفضليتها 25. (**ت**)–(**ث**) مراحل وسطية عند إدراج عقدة مفتاحها D وأفضليتها 9. (ج) الشجرة المكوَّمة بعد إدراج الجزأين (ت) و (ث). (ح) الشجرة المكوَّمة بعد إدراج عقدة مفتاحها F وأفضليتها 2.



الشكل 11.13 أعصاب شجرة بحث ثنائية. العصب الأيسر هو المظلل في (أ) والعصب الأيمن هو المظلل في (ب).

right spine هو المسار البسيط المؤلف من وصلات يمنى فقط انطلاقًا من الجذر. ***طول length*** العصب هو عدد العقد التي يحتويها.

ج. خذ الشجرة المكوَّمة T مباشرةً بعد أن يُدرج TREAP-INSERT العقدةَ x. وليكن C طول العصب الأيمن للشجرة الفرعية اليسرى لـ x. وليكن D هو طول العصب الأيسر للشجرة الفرعية اليمنى لـ x. أثبت أنَّ العدد الكلي للدورانات المنفذة خلال إدراج x يساوي $C + D$.

سنحسب الآن القيم المتوقعة لكل من C و D. نفترض، دون فقد العمومية، أنَّ المفاتيح هي $1, 2, \ldots, n$، لأننا نقارن أحدها بالآخر فقط.

ليكن $k = x.key$ و $i = y.key$ لكل عقدتين x و y، حيث $y \neq x$. نعرِّف متحولات عشوائية مؤشرة دليلية

$$X_{ik} = \mathrm{I}\{y \text{ is in the right spine of the left subtree of } x\} .$$

ح. بيِّن أنَّ $X_{ik} = 1$ إذا وفقط إذا كانت $y.key < x.key$، و $y.priority > x.priority$ وأن $y.priority > z.priority$ لجميع z التي تحقق $y.key < z.key < x.key$.

خ. أثبت أنَّ

$$\begin{aligned} \Pr\{X_{ik} = 1\} &= \frac{(k-i-1)!}{(k-i+1)!} \\ &= \frac{1}{(k-i+1)(k-i)} . \end{aligned}$$

د. أثبت أنَّ

$$\begin{aligned} \mathrm{E}[C] &= \sum_{j=1}^{k-1} \frac{1}{j(j+1)} \\ &= 1 - \frac{1}{k} . \end{aligned}$$

ذ. استخدم حجة التناظر لتبين أنَّ

$$\mathrm{E}[D] = 1 - \frac{1}{n-k+1} .$$

ر. استنتج أنَّ العدد المتوقَّع للدورانات المنفذة عند إدراج عقدة في شجرة مكوَّمة أصغر من 2.

ملاحظات الفصل

تعود فكرة توازن شجرة بحث إلى Adel'son-Vel'ski و Landis [2]، اللذين أدخلا صنفًا من أشجار البحث المتوازنة المسمَّاة "أشجار AVL" في عام 1962، والموصَّفة في المسألة 13-3. ثم أدخل J. E. Hopcroft صنفًا آخر من أشجار البحث عام 1970 يُسمى "أشجار 3-2" (غير منشورة). تحافظ أشجار 3-2 على التوازن بتناول درجات العقد في الشجرة. يدرس الفصل الثامن عشر تعميمًا للأشجار 3-2 ابتدعه باير وماكريت Bayer و McCreight [35] يسمى الأشجار المعمَّمة B-trees.

ابتكر باير Bayer [34] الأشجار الحمراء-السوداء تحت اسم "الأشجار المعمَّمة الثنائية المتناظرة" "symmetric binary B-trees". ودرس غويباس وسيدغويك Guibas و Sedgewick [155] خصائصها مطولاً وأدخلا اصطلاح اللون أحمر/أسود. في حين أعطى أندرسون Andersson [15] نوعًا آخر من الأشجار الحمراء-السوداء أكثر سهولةً في البرمجة. يسمِّي ڤايس Weiss [351] هذا النوع AA-trees. وهو يشبه الأشجار الحمراء-السوداء إلا أنَّ الأبناء اليُسْر قد لا يكونون حمرًا أبدًا.

اقترح سيدل وأراغون Seidel و Aragon [309] الأشجار المكوَّمة treaps، موضوع المسألة 13-4، وهي التنجيز المغتفل لمعجم في LEDA [253]، وهي مجموعة منجَّزة جيدًا من بنى المعطيات والخوارزميات.

وهناك أنواع أخرى كثيرة من أشجار البحث المتوازنة، منها الأشجار المتوازنة في الثقل [264]، والأشجار ذات الـ k جارًا k-neighbor trees [245]، وأشجار scapegoat [127]. ولعلَّ أكثرها إثارةً للاهتمام أشجار سبلاي "splay trees" التي أدخلها سليتور وتارجان Sleator و Tarjan [320]، وهي ذاتية التصحيح (انظر Tarjan [330] للوقوف على توصيف جيد لهذه الأشجار). تحافظ أشجار سبلاي على التوازن دون شرط توازن صريح مثل اللون. بل إنَّ العمليات (التي تتضمن دورانات) تُنفَّذ ضمن الشجرة كلما جرى النفاذ إليها. التكلفة المخمَّدة (انظر الفصل السابع عشر) لكل عملية على شجرة ذات n عقدة هي $O(\lg n)$.

توفِّر لوائح سكيب Skip lists [286] بديلاً للأشجار الثنائية المتوازنة. وهي لوائح مترابطة مزوَّدة بعدد من المؤشرات الإضافية. تُنفَّذ كل عملية معجمية على لائحة سكيب من n عنصرًا في زمن $O(\lg n)$.

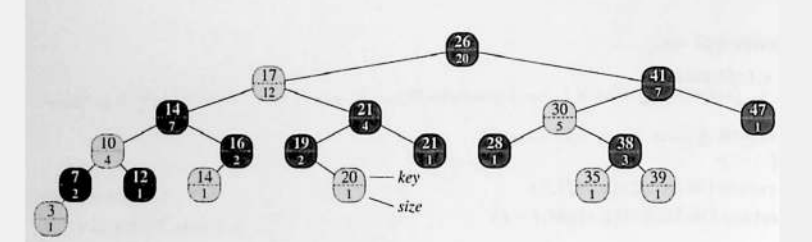
14 إغناء بنى المعطيات

لا تحتاج بعض الحالات الهندسية إلى أكثر من بنية معطيات في "كتاب مدرسي" — من مثل لائحةٍ مضاعَفة الترابط، أو جدول تلبيد، أو شجرة بحث ثنائية — غير أن العديد من الحالات الأخرى تحتاج إلى شيء من الإبداع. ومع ذلك، قد تحتاج في حالات نادرة إلى ابتكار نمط جديد بالكامل من بنى المعطيات. أما في الأغلب الأعم، فيكفي أن نغني بنى المعطيات الموجودة في الكتب الدراسية بتزويدها بمعلومات إضافية. ويمكنك بعدها أن تبرمج عملياتٍ جديدة على بنى المعطيات لدعم التطبيق المرغوب فيه. على أن إغناء بنى المعطيات لا يكون مباشرًا دومًا، بل يجب أن تُجرى العمليات التقليدية على بنى المعطيات بتحديث المعلومات المضافة والمحافظة عليها.

يناقش هذا الفصل بنيتَيْ معطيات أنشأناهما بإغناء الأشجار الحمراء-السوداء. يصف المقطع 1.14 بنية معطيات تدعم عمليات إحصائيات الترتيب العامة في مجموعة ديناميكية، بحيث يمكننا أن نجد بسرعة العدد الأصغر ذا الترتيب i في مجموعة، أو مرتبة عنصر معطى ضمن الترتيب الكلي لعناصر المجموعة. ويلخّص المقطع 2.14 إجرائية إغناء بنية معطيات، ويقدّم مبرهنة يمكنها تبسيط عملية إغناء الأشجار الحمراء-السوداء. ويستخدم المقطع 3.14 هذه المبرهنة للمساعدة في تصميم بنية معطيات للمحافظة على مجموعة ديناميكية من المجالات، كالمجالات الزمنية. فإذا كان لدينا مجال نريد الاستعلام عنه، يمكننا عندئذٍ إيجاد المجال الذي يتراكب overlap معه في المجموعة وبسرعة.

1.14 إحصائيات الترتيب الديناميكية

مهّد الفصل 9 لمفهوم إحصائية الترتيب. وبالأخص، فإن إحصائية الترتيب من الدرجة في مجموعة من n عنصرًا حيث $i \in \{1,2,\ldots,n\}$ هي ببساطة عنصر المجموعة الذي يمتلك المفتاح الأصغر ذا الترتيب i. وقد رأينا كيف نحدِّد أية إحصائية ترتيب في زمن $O(n)$ انطلاقًا من مجموعة غير مرتبة. وسنرى في هذا المقطع كيف يمكن تعديل الأشجار الحمراء-السوداء بحيث نستطيع تحديد أي إحصائية ترتيب لمجموعة ديناميكية في زمن $O(\lg n)$، وسنرى كذلك كيف يمكن حساب ***مرتبة rank*** عنصر – أي موقعه في الترتيب الخطي للمجموعة – في زمن $O(\lg n)$.



الشكل 1.14 شجرة إحصائيات الترتيب، وهي شجرة حمراء-سوداء مُغْنَاة. العقد المظللة حمراء والعقد الغامقة سوداء. إضافة إلى الواصفات المألوفة في كل عقدة x، جرت إضافة واصفة $x.size$ وهي تعبر عن عدد العقد-عدا الحارس- في الشجرة الفرعية التي جذرها x.

يبين الشكل 1.14 بنية معطيات تستطيع أن تدعم عمليات إحصائيات الترتيب السريعة. وتعرَّف ***شجرة إحصائيات الترتيب*** ***order-statistic tree T*** بأنها شجرة حمراء-سوداء مزوّدة بمعلومات إضافية مخزنة في كل عقدة. فإلى جانب الواصفات المألوفة في الأشجار الحمراء-السوداء وهي $x.key$ و $x.color$ و $x.p$ و $x.left$ و $x.right$ الموجودة في كل عقدة x، لدينا الواصفة $x.size$. تتضمن هذا الواصفة عدد العقد (الداخلية) في الشجرة الفرعية التي جذرها x (ومنها x نفسها)، أي أنها تتضمن حجم الشجرة الفرعية. فإذا عرّفنا حجم الحارس بأنه 0، أي إذا جعلنا $T.nil.size$ مساوية 0، عندها تكون لدينا المساواة:

$$x.size = x.left.size + x.right.size + 1 \ .$$

ليس من الضروري أن تكون المفاتيح متمايزة في شجرة إحصائيات الترتيب (فالشجرة الواردة في الشكل 1.14 مثلاً، لها مفتاحان قيمتهما 14 ومفتاحان قيمتهما 21). في حال وجود مفاتيح متساوية لا يكون مفهوم المرتبة المذكور آنفًا معرّفًا تعريفًا جيدًا، ونزيل هذا الالتباس في شجرة إحصائيات الترتيب بتعريف مرتبة عنصر على أنها الموقع الذي سيظهر فيه فيما لو طبعنا الشجرة بتجوال بيني inorder walk. في الشكل 1.14 مثلاً، تكون مرتبة المفتاح 14 المخزن في عقدة سوداء هي 5، وتكون مرتبة المفتاح 14 المخزن في عقدة حمراء هي 6.

استحضار عنصر ذي مرتبة معطاة

قبل أن نبين كيف نحافظ على معلومة الحجم هذه أثناء الإدراج والحذف، لنتفحّص تنجيز استعلامَيْن لإحصائيات الترتيب يَستخدمان هذه المعلومة الإضافية. نبدأ بعمليةٍ تستحضر عنصرًا ذا مرتبة معطاة. يُعيد الإجراء $\text{OS-SELECT}(x, i)$ مؤشرًا إلى العقدة التي تحتوي المفتاح الأصغر ذا الترتيب i في الشجرة الفرعية التي رأسها x. ولإيجاد العقدة ذات المفتاح الأصغر ذي الترتيب i في شجرة إحصائيات الترتيب T نستدعي $\text{OS-SELECT}(T.root, i)$.

```
OS-SELECT(x, i)
1  r = x.left.size + 1
2  if i == r
3      return x
4  elseif i < r
5      return OS-SELECT(x.left, i)
6  else return OS-SELECT(x.right, i - r)
```

في السطر 1 من OS-SELECT نحسب r وهو مرتبة العقدة x في الشجرة الفرعية التي جذرها x. إن قيمة $x.left.size$ هي عدد العقد التي ترِد قبل x في تجوال بيني للشجرة الفرعية التي جذرها x. وهكذا يكون $x.left.size + 1$ مرتبة x في الشجرة الفرعية التي جذرها x. إذا كان $i = r$ فإن العقدة x تكون العنصر الأصغر ذا الترتيب i، وعليه فإننا نعيد x في السطر 3. وإذا كان $i < r$ يكون العنصر الأصغر ذو الترتيب i موجودًا في الشجرة الفرعية اليسرى لـ x، لذا نطبق العودية على $x.left$ في السطر 5. وإذا كان $i > r$، يكون العنصر الأصغر ذو الترتيب i موجودًا في الشجرة الفرعية اليُمنى لـ x. ولما كانت الشجرةُ الفرعية التي جذرها x تحتوي r عنصرًا ترِد قبل الشجرة الفرعية اليمنى لـ x في التجوال البيني للشجرة، فإن العنصر الأصغر ذا الترتيب i في الشجرة الفرعية التي جذرها x يكون هو العنصر الأصغر ذا الترتيب $(i - r)$ في الشجرة الفرعية التي جذرها $x.right$. يحدَّد هذا العنصر عوديًّا في السطر 6.

ولمعاينة كيفية عمل OS-SELECT، نبحث عن العنصر الأصغر ذي الترتيب 17 في شجرة إحصائيات الترتيب الواردة في الشكل 1.14. نبدأ بالجذر x الذي يحمل المفتاح 26 ولدينا قيمة $i = 17$. ولما كان حجم الشجرة الفرعية اليسرى لـ 26 هو 12، فإن مرتبتها هي 13. وهكذا، نعلم أن العقدة ذات المرتبة 17 هي $17 - 13 = 4$ أي إننا نبحث عن العنصر الرابع في الصغر في الشجرة الفرعية اليمنى لـ 26. بعد الاستدعاء العودي تصبح x هي العقدة ذات المفتاح 41 و $i = 4$. ولما كان حجم الشجرة الفرعية اليسرى لـ 41 هو 5، فإن رتبته في شجرته الفرعية هي 6. وهكذا نعلم أن العقدة ذات المرتبة 4 هي العنصر الرابع في الصغر في الشجرة الفرعية اليسرى لـ 41. وبعد الاستدعاء العودي تصبح x هي العقدة ذات المفتاح 30 ومرتبتها في شجرتها الفرعية هو 2. ومن ثَمَّ، نطبق العودية مرة ثانية لنجد العنصر الأصغر الثاني $(4 - 2 = 2)$ في الشجرة الفرعية التي جذرها هو العقدة ذات المفتاح 38. هنا، نجد أن حجم الشجرة الفرعية اليسرى لهذه العقدة هو 1، وهذا يعني أنها العنصر الثاني في الصغر. وهكذا يعيد الإجراء مؤشرًا إلى العقدة ذات المفتاح 38 .

ولما كان كل استدعاء عودي يقودنا في كل مرة إلى مستوى أعمق واحد داخل شجرة إحصائيات الترتيب، فإن الزمن الإجمالي لـ OS-SELECT يتناسب في أسوأ الحالات مع ارتفاع الشجرة. وحيث إن الشجرة هي شجرة حمراء-سوداء، فإن ارتفاعها هو $O(\lg n)$ حيث n هو عدد العقد. وهكذا يكون زمن تنفيذ OS-SELECT هو $O(\lg n)$ في حالة مجموعة ديناميكية ذات n عنصرًا.

تحديد مرتبة عنصر

ليكن لدينا مؤشر إلى عقدة x في شجرة إحصائيات الترتيب T. يعيد الإجراءُ OS-RANK موقعَ x في الترتيب الخطي الذي يحدده التجوال البيني للشجرة T.

```
OS-RANK(T, x)
1  r = x.left.size + 1
2  y = x
3  while y ≠ T.root
4       if y == y.p.right
5            r = r + y.p.left.size + 1
6       y = y.p
7  return r
```

يعمل الإجراء كما يلي: يمكن النظر إلى مرتبة x على أنها عدد العقد التي تسبق x في تجوالٍ بَيْنِيٍّ للشجرة مضافًا إليه 1 لـ x نفسه. يحافظ OS-RANK على لامتغير الحلقة التالي:

عند بداية كل تكرار للحلقة **while** في الأسطر 3-6، يكون r هو مرتبة $x.key$ في الشجرة الفرعية التي جذرها العقدة y.

نستخدم لامتغير الحلقة هذا لنبيِّن أن OS-RANK يعمل بوجهٍ صحيح كما يلي:

الاستبداء: قبل التكرار الأول، يضع السطر الأول في r مرتبة $x.key$ في الشجرة الفرعية التي جذرها x. إن وضع $y = x$ في السطر 2 يجعل لامتغير الحلقة صحيحًا عندما ينفذ الاختبار في السطر 3 أول مرة.

المحافظة: في نهاية كل تكرار للحلقة **while**، نجعل $y = y.p$. لذا، يجب أن نبين أنه إذا كان r هو مرتبة $x.key$ في الشجرة الفرعية التي جذرها y في بداية جسم الحلقة، فإن r يكون مرتبة $x.key$ في الشجرة الفرعية التي جذرها $y.p$ في نهاية جسم الحلقة. وفي كل تكرار للحلقة **while** نهتم بالشجرة الفرعية التي جذرها $y.p$. وقد سبق أن حسبنا عدد العقد في الشجرة الفرعية التي جذرها هو العقدة y التي تسبق x في التجوال البيني، لذا، يجب إضافة العقد في الشجرة الفرعية التي جذرها هو أخ y الذي يسبق x في التجوال البيني ويضاف 1 أيضًا لـ $y.p$ إذا كان هو بدوره يسبق x. أما إذا كان y ابنًا أيسر، عندها لا يمكن لـ $y.p$ ولا لأية عقدة في الشجرة الفرعية اليمنى لـ $y.p$ أن تسبق العقدة x. عندها نترك r وحدها. وفيما عدا ذلك، فإن y ابن أيمن، وجميع العقد في الشجرة الفرعية اليسرى لـ $y.p$ تسبق x، كما هو حال $y.p$ نفسه. لذا، نضيف $y.p.left.size + 1$، في السطر 5، إلى القيمة الحالية لـ r .

الانتهاء: تنتهي الحلقة عندما تصبح $y = T.root$ بحيث تكون الشجرة الفرعية التي جذرها y هي كامل الشجرة. لذا، تكون قيمة r هي مرتبة $x.key$ في كامل الشجرة.

مثلاً، إذا نفذنا OS-RANK على شجرة إحصائيات الترتيب في الشكل 1.14 للبحث عن مرتبة العقدة ذات المفتاح 38، حصلنا على المتتالية اللاحقة من قيم $y.key$ و r في بداية الحلقة **while**:

التكرار	$y.key$	r
1	38	2
2	30	4
3	41	4
4	26	17

يعيد الإجراءُ المرتبةَ 17.

ولما كان كل تكرار للحلقة **while** يستغرق $O(1)$ من الزمن، و y ترتفع نحو الأعلى بمقدار مستوى واحد في الشجرة مع كل تكرار، فإن زمن تنفيذ OS-RANK يتناسب في أسوأ الحالات مع ارتفاع الشجرة: $O(\lg n)$ في شجرة إحصائيات الترتيب ذات n عقدة.

المحافظة على أحجام الأشجار الفرعية

إذا عُلِمت قيمةُ الواصفة $size$ في كل عقدة، استطاع OS-SELECT و OS-RANK حساب معلومات إحصائية الترتيب بسرعة. ولكن إذا لم تستطع عملياتُ التعديل الأساسية على الأشجار الحمراء–السوداء المحافظةَ على هذه الواصفات محافظةً فعّالة فستذهب جهودنا سُدًى. وسنبين الآن أن عمليتي الإدراج والحذف تحافظان على حجوم الأشجار الفرعية دون التأثير في الزمن المقارب لتنفيذ أيٍّ من هاتين العمليتين.

لاحظنا في المقطع 3.13 أن الإدراج في شجرة حمراء–سوداء يتألف من مرحلتين. تعتمد المرحلة الأولى على النُّزول في الشجرة اعتبارًا من الجذر، وإدراج العقدة الجديدة كابن لعقدة موجودة. في حين تقوم المرحلة الثانية بالصعود إلى الأعلى في الشجرة، وتغيير الألوان وإجراء الدورانات اللازمة للمحافظة على الخصائص الحمراء–السوداء.

وللمحافظة على حجوم الأشجار الفرعية في المرحلة الأولى، ما علينا إلا إضافة 1 إلى $x.size$ في كل عقدة x على المسار البسيط الممتد من الجذر نزولاً نحو الأوراق. تحصل العقدة المضافة على قيمة 1 في واصفة $size$ الخاصة بها. ولما كان ثمة $O(\lg n)$ عقدة على المسار المذكور، فإن التكلفة الإضافية للمحافظة على الواصفات $size$ تكون $O(\lg n)$.

أما في المرحلة الثانية، فتتمثل التغييرات البنيوية الوحيدة على الشجرة الحمراء–السوداء الأساسية في الدورانات التي يحدث منها على الأكثر اثنان. وكذلك فإن الدوران هو عملية محلية: إذ إن عقدتَيْن فقط تصبح واصفة $size$ فيهما غير صحيحة. والوصلة التي ينفَّذ حولها الدوران تتوقف على هاتين العقدتين. وبالرجوع إلى رماز LEFT-ROTATE(T, x) الوارد في المقطع 2.13 نضيف السطرين التاليين:



الشكل 2.14 تحديث أحجام الأشجار الفرعية أثناء الدورانات. إن الوصلة التي يجري الدوران حولها منوطة بالعقدتين اللتين يجب تحديث واصفات *size* فيهما. وإن عمليات التحديث محلية، وتحتاج فقط إلى المعلومة *size* المخزنة في x و y، وإلى جذور الأشجار الفرعية الممثلة على شكل مثلثات.

```
13   y.size = x.size
14   x.size = x.left.size + x.right.size + 1
```

يوضح الشكل 2.14 كيف تُحدَّث هاتان الواصفتان. ويكون التغيير في RIGHT-ROTATE مناظرًا لهذا التغيير.

وحيث إنه سيُنفَّذ دورانان على الأكثر أثناء الإدراج في شجرة حمراء–سوداء، فإننا نحتاج إلى زمنٍ إضافي $O(1)$ فقط كي نحدِّث الواصفات *size* في المرحلة الثانية. وهكذا، يكون الزمن الإجمالي للإدراج في شجرة إحصائية الترتيب ذات n عقدة هو $O(\lg n)$، وهو مقارب للإدراج في شجرة حمراء–سوداء عادية.

وبالمثل، يتألف الحذف من شجرة حمراء–سوداء مرحلتين: المرحلة الأولى تعمل على شجرة البحث الأساسية، والثانية تتسبب في ثلاثة دورانات على الأكثر، وفيما عدا ذلك فهي لا تقوم بأية تغييرات بنيوية (انظر المقطع 4.13). فالمرحلة الأولى إما أن تَحذف عقدةً واحدة y من الشجرة، وإما أن تنتقل إلى موقعٍ أعلى منها ضمن الشجرة. ولتحديث حجوم الأشجار الفرعية نقوم ببساطة بعبور المسار من العقدة y (اعتبارًا من موضعها الأصلي في الشجرة) إلى الأعلى باتجاه الجذر، وننقص 1 من الواصفة *size* لكل عقدة على المسار. ولما كان طول هذا المسار هو $O(\lg n)$ في حالة شجرة حمراء–سوداء ذات n عقدة، فإن الزمن الإضافي اللازم للمحافظة على الواصفات *size* في المرحلة الأولى هو $O(\lg n)$. أما الدورانات ذات الزمن $O(1)$ في المرحلة الثانية من الحذف، فيمكن معالجتها بالطريقة نفسها كما في الإدراج. وهكذا، تستغرق عمليتا الإدراج والحذف، وما يتضمنانه من محافظة على الواصفات *size*، زمنًا $O(\lg n)$ في حالة شجرة إحصائيات الترتيب ذات n عقدة.

تمارين

1-1.14

بيّن كيف تعمل $\text{OS-SELECT}(T.root, 10)$ على الشجرة الحمراء–السوداء T التي مرت معنا في الشكل 1.14.

2-1.14
بيّن كيف تعمل $\text{OS-RANK}(T,x)$ على الشجرة الحمراء-السوداء T التي مرت معنا في الشكل 1.14 والعقدة x حيث $x.key = 35$.

3-1.14
اكتب نسخة غير عودية من OS-SELECT.

4-1.14
اكتب إجراءً عوديًا $\text{OS-KEY-RANK}(T,k)$ يتخذ شجرةً إحصائية الترتيب T ومفتاحًا k دخلاً له، ويعيد مرتبة k في المجموعة الديناميكية الممثلة بـ T. افترض أن مفاتيح T متمايزة.

5-1.14
لدينا عنصر x معطىً في شجرة إحصائيات الترتيب ذات n عقدة وعدد طبيعي i. كيف يمكننا تحديد العنصر اللاحق لـ x ذي الترتيب i في الترتيب الخطي للشجرة وذلك خلال زمن $O(\lg n)$؟

6-1.14
لاحظ أنه كلما أشرنا إلى الواصفة *size* لعقدة ما، سواء في OS-SELECT أو في OS-RANK، فإننا نستعملها لحساب مرتبة عقدة فقط. بناءً على ذلك، افترض أننا نخزن في كل عقدة مرتبتها في الشجرة الفرعية التي تمثل العقدة جذرًا لها. بيّن كيف يمكن المحافظة على هذه المعلومة أثناء الإدراج والحذف. (تذكّر أن هاتين العمليتين قد تسبّبان دورانات.)

7-1.14
بيّن كيف تُستخدم شجرة إحصائية الترتيب لإحصاء عدد عمليات القَلْب (انظر المسألة 2-4) في مصفوفة طولها n في زمن $O(n\lg n)$.

★ *8-1.14*
ليكن لدينا n وترًا على دائرة بحيث يُعرَّف كل وتر بنقطتي الطرفين. صِفْ خوارزميةً مقدار زمنها $O(n\lg n)$ لتحديد عدد أزواج الأوتار التي تتقاطع داخل الدائرة. (مثلاً إذا كانت جميع الأوتار التي عددها n أقطارًا تلتقي في المركز، فعندها يكون الجواب الصحيح $\binom{n}{2}$.) افترض أنه لا يوجد وتران يشتركان في إحدى النهايتين.

2.14 كيف نغني بنية معطيات

إن الغرض من عملية إغناء بنية معطيات أساسية هو تعزيز فعاليةٍ إضافية، وتَحدث هذه العملية حدوثًا متكررًا عند تصميم الخوارزميات. سنستخدم هذه العملية أيضًا في المقطع التالي لتصميم بنية معطيات تدعم العمليات على المجالات. وسندرس في هذا المقطع الخطوات المتبّعة في عملية الإغناء، وسنبرهن أيضًا نظريةً تسمح لنا بإغناء الأشجار الحمراء-السوداء بسهولة في حالاتٍ كثيرة.

يمكن تقسيم عملية إغناء بنية معطيات إلى أربع خطوات:

1. اختيار بنية معطيات أساسية.
2. تحديد المعلومات الإضافية المراد المحافظة عليها في بنية المعطيات الأساسية.
3. التحقق من إمكان المحافظة على المعلومات الإضافية في عمليات التعديل الأساسية على بنية المعطيات الأساسية.
4. استحداث عمليات جديدة.

وكما في أية طريقة تصميم منهجية، لا يفترض بك أن تتبع الخطوات وفق الترتيب المعطى اتباعًا أعمى؛ فمعظم أعمال التصميم تقوم على عنصر المحاولة والخطأ، ويحدث التطور في جميع الخطوات على التوازي عادةً. فمثلاً، من العبث تحديدُ معلوماتٍ إضافية وابتكار عمليات جديدة (المرحلتان 2 و 4) إذا لم تكن لدينا القدرة على المحافظة على المعلومات الإضافية محافظةً فعالة. على كل حال، فإن هذه الطريقة ذات الخطوات الأربع من شأنها أن توجّه جهودك توجيهًا جيدًا في إغناء بنى المعطيات، وهي في الوقت نفسه طريقة جيدة لتنظيم التوثيق المتعلق ببنية المعطيات المُغناة.

وقد اتّبعنا هذه الخطوات في المقطع 1.14 في تصميم أشجار إحصائيات الترتيب. ففي الخطوة الأولى اخترنا أشجارًا حمراء-سوداء كبنية معطيات أساسية. ولعل السر في صلاحية الأشجار الحمراء-السوداء يكمن في دعمها الفعّال لعمليات المجموعات الديناميكية الأخرى المتعلقة بالترتيب الكلي مثل MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR.

وفي الخطوة 2، أضفنا الواصفة *size* التي تخزّن فيها كلُّ عقدة x حجمَ الشجرة الفرعية التي جذرها x. وترمي المعلومة الإضافية عمومًا إلى جعل العمليات أكثر فعالية. فمثلاً، كان من الممكن تنجيز OS-SELECT و OS-RANK باستخدام المفاتيح المخزنة في الشجرة فقط، ولكننا لم نكن لنحصل على زمن التنفيذ $O(\lg n)$. وفي بعض الأحيان تكون المعلومة الإضافية على شكل مؤشر على معلومات بدلاً من كونها معطيات كما في التمرين 14.2-1.

وفي الخطوة 3، تأكّدنا أن الإدراج والحذف قد يحافظان على الواصفات *size* مع بقاء التنفيذ ضمن زمن $O(\lg n)$. في الحالة المثالية، نحتاج فقط إلى تحديث عدد قليل من عناصر بنية المعطيات بهدف المحافظة على المعلومة الإضافية. فمثلاً، لو خزّنّا ضمن كل عقدة في الشجرة مرتبتَها، لنُفّذ الإجراءان OS-SELECT و OS-RANK بسرعة، ولكن إدراج عنصر أصغري جديد قد يتسبب في تغيير هذه المعلومة في كل عقدة من الشجرة. أما لو قمنا بدلاً من ذلك بتخزين حجوم الأشجار الفرعية، لأحدَثَ إدراجُ عنصر جديد تغيُّرًا في المعلومات في $O(\lg n)$ عقدة فقط.

في الخطوة 4، استحدثنا العمليتين OS-SELECT و OS-RANK. وواقع الأمر أن الحاجة إلى عمليات

جديدة كانت هي الدافع لنا للاهتمام بإغناء بنية المعطيات بالدرجة الأولى، علمًا بأننا نستعمل بين حينٍ وآخر المعلومةَ الإضافية لتسريع العمليات الموجودة بدلاً من تطوير عمليات جديدة كما في التمرين 14.2-1.

إغناء الأشجار الحمراء–السوداء

عندما تكون الأشجار الحمراء–السوداء هي البنية الأساسية لبنية معطيات مُغناة، يمكننا عندها البرهان على أن الإدراج والحذف يستطيعان المحافظة على بعض أنواع المعلومات الإضافية محافظةً فعالةً، وهذا ما يجعل الخطوة 3 سهلة جدًا. ويشبه برهانُ النظريةِ التاليةِ المناقشةَ التي وردت معنا في المقطع 14.1، وهي أنه يمكن المحافظة على الواصفة $size$ في أشجار إحصائيات الترتيب.

نظرية 14.1 *(إغناء شجرة حمراء–سوداء)*

لتكن f واصفةً تُغني شجرة حمراء–سوداء T ذات n عقدة، ونفترض أن قيمة f في كل عقدة x تعتمد فقط على المعلومات الموجودة في العقد x و $x.left$ و $x.right$، وقد تتضمن $x.left.f$ و $x.right.f$. عندها، يمكن المحافظة على قيم f في جميع عقد T أثناء الإدراج والحذف دون التأثير بطريقة مقاربة على الأداء $O(\lg n)$ لهذه العمليات.

البرهان تقوم الفكرة الرئيسة في البرهان على أن التغيير في واصفة f في عقدة x ينتقل إلى أسلاف x فقط في الشجرة، بمعنى أن تغيير $x.f$ قد يضطرنا إلى تحديث $x.p.f$ فقط دون سواه؛ وقد يضطرنا تحديث $x.p.f$ بدوره إلى تحديث $x.p.p.f$ فقط دون سواه؛ وهكذا صعودًا إلى الأعلى في الشجرة. وعندما نُحدِّث $T.root.f$، فإنه لا توجد عقدة أخرى تتعلق بالقيمة الجديدة، لذا تتوقف العملية. ولما كان ارتفاع الشجرة الحمراء–السوداء هو $O(\lg n)$، فإن تغيير واصفة f في عقدة يكلّفنا $O(\lg n)$ لتحديث جميع العقد المعتمدة على هذا التغيير.

يتألف إدراج عقدة x في T من مرحلتين. (انظر المقطع 13.3.) تُدرج المرحلة الأولى x باعتباره ابنًا لعقدة موجودة $x.p$. ويمكننا حساب $x.f$ في زمن $O(1)$ لأنه –حسب فرضنا– يتعلق فقط بالمعلومات الموجودة في الواصفات الأخرى لـ x نفسها وبالمعلومات الموجودة في أبناء x، إلا أن ابني x هما الحارس $T.nil$. بعد حساب $x.f$ ينتقل التغيير إلى الأعلى في الشجرة، لذا، يكون الزمن الإجمالي للمرحلة الأولى من الإدراج هو $O(\lg n)$. وفي أثناء المرحلة الثانية، تأتي التغييرات البنيوية الوحيدة على الشجرة من الدورانات. ولما كان الدوران يتسبب في التغيير على عقدتين فقط، فإن الزمن الإجمالي لتحديث الواصفات f هو $O(\lg n)$ لكل دوران. ولما كان عدد الدورانات أثناء الإدراج لا يتجاوز اثنين، فإن الزمن الإجمالي للإدراج يكون $O(\lg n)$.

وعلى غرار الإدراج، يتألف الحذف من مرحلتين. (انظر المقطع 13.4.) ففي المرحلة الأولى، تَحدث التغييرات على الشجرة عند إزالة العقدة المحذوفة من الشجرة. فإذا كان للعقدة المحذوفة ابنان في ذلك الوقت،

انتقل خَلَفُها إلى موضع العقدة المحذوفة. يكلِّف انتقال التحديثات على f التي تسببت بها هذه التغييرات $O(\lg n)$ على الأكثر، لأن التغييرات تؤثر على الشجرة محليًّا. ويتطلب تصليح الشجرة الحمراء-السوداء أثناء المرحلة الثانية ثلاثة دورانات على الأكثر، ويحتاج كل دوران إلى زمن $O(\lg n)$ على الأكثر لانتقال التحديثات على f. لذا، وكما في الإدراج، يكون الزمن الإجمالي للحذف $O(\lg n)$. ■

وفي حالات كثيرة، من مثل المحافظة على الواصفات *size* في أشجار إحصائيات الترتيب، تكون تكلفة التحديث بعد الدوران هي $O(1)$ بدلاً من $O(\lg n)$ المستمدة من برهان النظرية 1.14. يعطي التمرين 3-2.14 مثالاً عن ذلك.

تمارين

1-2.14

بيِّن كيف يمكننا، بإضافة المؤشرات إلى العقد، دعم كلٍّ من الاستعلامات MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR في المجموعات الديناميكية خلال زمن $O(1)$ في أسوأ الحالات في شجرة إحصائيات الترتيب المُغناة. ينبغي ألاَّ يتأثر الأداء المقارب لبقية العمليات المجراة على أشجار إحصائيات الترتيب.

2-2.14

هل يمكننا المحافظة على الارتفاعات السوداء للعقد في شجرة حمراء-سوداء على اعتبار أنها واصفات في عقد الشجرة دون التأثير في الأداء المقارب لأيٍّ من العمليات على الشجرة الحمراء-السوداء؟ بيِّن كيف يمكن ذلك أو أثبت لمَ لا يمكن ذلك. ماذا عن المحافظة على أعماق العقد؟

3-2.14 ⋆

ليكن ⊗ معاملاً اثنانيًّا تجميعيًّا، وليكن a واصفةً محفوظةً في كل عقدة من عقد شجرة حمراء-سوداء. ولنفترض أننا نريد أن نضمِّن في كل عقدة x واصفةً إضافيةً f بحيث يكون $x.f = x_1 \cdot a \otimes x_2 \cdot a \otimes \cdots \otimes x_m \cdot a$ حيث $x_1, x_2, \ldots, x_m$ هي السرد البيني للعقد في الشجرة الفرعية التي جذرها x. بيِّن كيف يمكن تحديث الواصفات f في زمن $O(1)$ بعد إجراء دوران. عدِّل برهانك قليلاً بحيث يمكن تطبيقه على الواصفات *size* في أشجار إحصائيات الترتيب.

4-2.14 ⋆

نرغب في إغناء أشجارٍ حمراء-سوداء بالعملية RB-ENUMERATE(x, a, b) التي تعطي خرجًا يتمثل في جميع المفاتيح k حيث $a \le k \le b$ في شجرةٍ حمراء-سوداء جذرها x. صِفْ كيف يمكن تنجيز RB-ENUMERATE في زمن $\Theta(m + \lg n)$، حيث m عدد المفاتيح في الخرج و n عدد العقد الداخلية في الشجرة. (*تلميح*: لا حاجة إلى إضافة واصفات جديدة إلى الشجرة الحمراء-السوداء.)

3.14 أشجار المجالات

في هذا المقطع، سنُغني الأشجار الحمراء-السوداء بحيث تدعم العمليات على المجموعات الديناميكية الخاصة بالمجالات. يعرَّف ***المجال المغلق closed interval*** بأنه زوج مرتَّب من الأعداد الحقيقية $[t_1, t_2]$ حيث $t_1 \leq t_2$. يمثِّل المجالُ $[t_1, t_2]$ المجموعةَ $\{t \in \mathbb{R} : t_1 \leq t \leq t_2\}$. أما المجالات ***المفتوحة open*** و ***نصف المفتوحة half-open*** فلا تتضمن كلتا النهايتين أو إحداهما في المجموعة على الترتيب. سنفترض في هذا المقطع أن المجالات مغلقة، وبذلك يكون توسيع النتائج إلى المجالات المفتوحة ونصف المفتوحة أمرًا مباشرًا وواضحًا على مستوى المفاهيم.

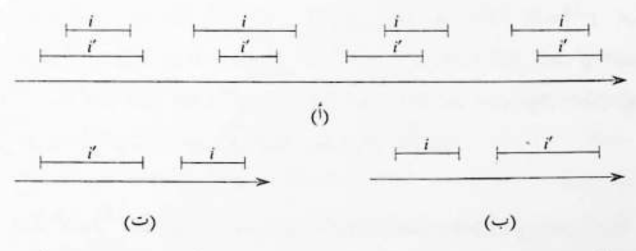
هذا وإن استعمال المجالات ملائمٌ لتمثيل الأحداث التي يَشغل كلٌّ منها مدةً مستمرة. فقد نرغب مثلاً بالاستعلام من قاعدة معطيات عن المجالات الزمنية لمعرفة الأحداث التي تجري أثناء مجال مُعطى. تقدِّم بنية المعطيات في هذا المقطع وسيلةً ناجعة للمحافظة على قاعدة معطيات المجالات هذه.

يمكن أن نمثل المجال $[t_1, t_2]$ كغرضٍ i مزوَّدٍ بالواصفتين $i.low = t_1$ (***النقطة الطرفية الدنيا low endpoint***) و $i.high = t_2$ (***النقطة الطرفية العليا high endpoint***). نقول إن المجالين i و i' ***متراكبان overlap*** إذا كان $i \cap i' \neq \emptyset$، بمعنى أن $i.low \leq i'.high$ و $i'.low \leq i.high$. وكما يبيِّن الشكل 3.14، يحقِّق أيُّ زوجٍ من المجالات i و i' خاصية ***التفرُّع الثلاثي للمجالات interval trichotomy*** بمعنى أنهما يحققان واحدة فقط من هذه الخصائص الثلاث التالية:

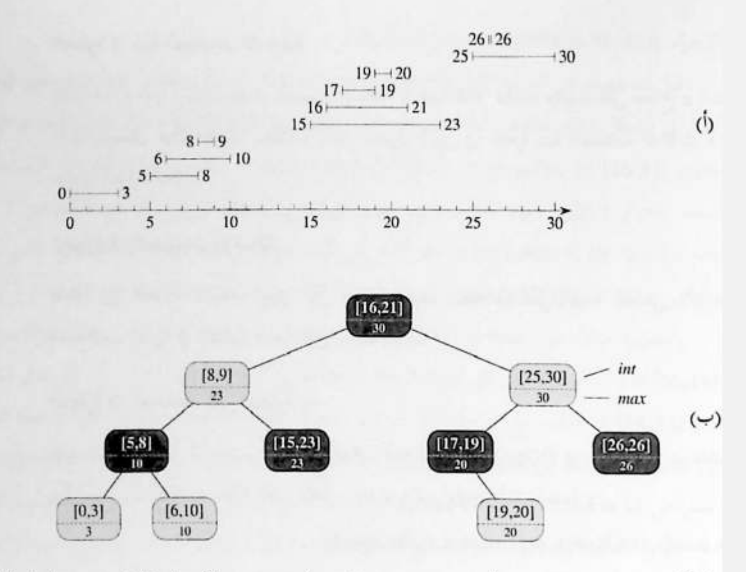
أ. i و i' يتراكبان

ب. يقع i إلى يسار i' (أي $i.high < i'.low$)

ت. يقع i إلى يمين i' (أي $i'.high < i.low$)



الشكل 3.14 خاصية التفرُّع الثلاثي للمجالات في حالة مجالَيْن مغلقين i و i'. (أ) إذا تَراكبَ i و i' فهناك أربع حالات يكون في كل منها $i.low \leq i'.high$ و $i'.low \leq i.high$. **(ب)** المجالان لا يتراكبان و $i.high < i'.low$. **(ت)** المجالان لا يتراكبان و $i'.high < i.low$.



الشكل 4.14 شجرة مجالات. (أ) مجموعة من 10 مجالات تظهر مرتبة من الأدنى إلى الأعلى حسب النقطة الطرفية اليسرى. (**ب**) الشجرة التي تمثل هذه المجالات. تحتوي كل عقدة x مجالاً يظهر على الشكل فوق الخط المنقط، والقيمة العظمى لأية نقطة طرفية للمجالات في الشجرة الفرعية التي جذرها x، وتظهر في الشكل تحت الخط المنقط. يسمح التجوال البيني للشجرة بسرد العقد مرتبةً حسب النقطة الطرفية اليسرى.

تعرّف ***شجرة المجال interval tree*** بأنها شجرة حمراء-سوداء تحافظ على مجموعة ديناميكية من العناصر التي يحتوي كل عنصر x فيها مجالاً $x.int$. تدعم أشجار المجال العمليات التالية:

INTERVAL-INSERT(T, x) وهي تضيف إلى شجرة المجال T العنصر x الذي يفترض أن تحتوي الواصفة int فيه مجالاً.

INTERVAL-DELETE(T, x) وهي تحذف العنصر x من شجرة المجال T.

INTERVAL-SEARCH(T, i) وهي تعيد مؤشرًا إلى العنصر x الموجود في شجرة المجال T بحيث يتراكب مجاله $x.int$ مع المجال i، أو تعيد مؤشّرًا إلى الحارس $T.nil$ إن لم يوجد هذا العنصر في المجموعة.

يبين الشكل 4.14 كيف تمثل شجرةُ المجال مجموعةً من المجالات. سنتتبع الخطوات الأربع في الطريقة التي وصفناها في المقطع 2.14 في سياق مراجعتنا لتصميم شجرة المجال والعمليات التي تنفّذ عليها.

الخطوة 1: بنية المعطيات الأساسية

نختار شجرة حمراء-سوداء بحيث تتضمن كلُّ عقدة x فيها مجالاً $x.int$ وبحيث يمثِّل مفتاحُ x النقطةَ الطرفيةَ الدنيا للمجال $x.int.low$. وهكذا، يسرد التجوالُ البيني في شجرة بنية المعطيات المجالاتِ مرتبةً حسب نقاطها الطرفية الدنيا.

الخطوة 2: المعلومات الإضافية

إضافة إلى المجالات نفسها، تحتوي كلُّ عقدة x قيمةً $x.max$، تمثِّل القيمةَ العظمى لأية نقطةٍ طرفيةٍ للمجالات المخزنة في الشجرة الفرعية التي جذرها x.

الخطوة 3: المحافظة على المعلومات

يجب أن نتحقق من أن عمليتي الإدراج والحذف تُنْجَزان في زمن $O(\lg n)$ في شجرة مجالات ذات n عقدة. يمكن أن نحدد $x.max$ بدلالة المجال المعطى $x.int$ والقيم max لأبناء العقدة x:

$$x.max = \max(x.int.high, x.left.max, x.right.max) .$$

إذن، وبحسب النظرية 1.14 ينفَّذ كل من الإدراج والحذف في زمن $O(\lg n)$. والواقع أنه بإمكاننا تحديث الواصفات max بعد إجراء دوران خلال زمن $O(1)$، كما يبين التمرينان 14.2-3 و 14.3-1 .

الخطوة 4: استحداث عمليات جديدة

إن العملية الجديدة الوحيدة التي نحتاج إليها هي $\text{INTERVAL-SEARCH}(T, i)$ التي تبحث عن عقدة في شجرة T بحيث يتراكب مجالها مع المجال i. إذا لم تجد هذه العملية أي مجال يتراكب مع i في الشجرة فإنها تعيد مؤشرًا للحارس $T.nil$.

```
INTERVAL-SEARCH(T, i)
1  x = T.root
2  while x ≠ T.nil and i does not overlap x.int
3      if x.left ≠ T.nil and x.left.max ≥ i.low
4          x = x.left
5      else x = x.right
6  return x
```

يبدأ البحث عن مجال يتراكب مع i انطلاقًا من x باعتبارها جذرًا للشجرة ويستمر نزولاً. ويتوقف البحث عندما يجد مجالاً متراكبًا، أو عندما تشير x إلى الحارس $T.nil$. ونظرًا إلى أن كل تكرار من الحلقة الرئيسية يستغرق $O(1)$ من الزمن، ونظرًا إلى أن ارتفاع شجرة حمراء-سوداء ذات n عقدة هو $O(\lg n)$، فإن

الإجراء INTERVAL-SEARCH يستغرق زمنًا $O(\lg n)$.

قبل أن نفصح عن السبب الذي يجعل INTERVAL-SEARCH صحيحًا، لندرس كيفية عمله على شجرة المجال في الشكل 4.14. لنفترض أننا نبحث عن مجال يتراكب مع المجال $i = [22, 25]$. نبدأ بالجذر x الذي يحتوي $[16, 21]$ ولا يتراكب مع i. ولما كان $x.left.max = 23$ أكبر من $i.low = 22$، فإن الحلقة تستمر باعتبار x الابن الأيسر للجذر-وهي العقدة التي تحتوي $[8, 9]$، الذي لا يتراكب كذلك مع i. في هذه المرة لدينا $x.left.max = 10$ وهو أصغر من $i.low = 22$، تستمر الحلقة إذًا مع الابن الأيمن لـ x بدلاً من x. إن المجال $[15, 23]$ المخزن في هذه العقدة يتراكب مع i، لذا يعيد الإجراءُ هذه العقدة.

ولنضرب مثالاً على عملية بحث غير ناجحة. لنفترض أننا نرغب بالبحث عن مجال يتراكب مع $i = [11, 14]$ في شجرة المجال التي الواردة في الشكل 4.14. نبدأ هنا أيضًا بالجذر x. ولما كان مجال الجذر وهو $[16, 21]$ لا يتراكب مع i، ونظرًا إلى أن $x.left.max = 23$ أكبر من $i.low = 11$، فإننا نتجه إلى اليسار نحو العقدة التي تحتوي المجال $[8, 9]$. إن المجال $[8, 9]$ لا يتراكب مع i و $x.left.max = 10$ أصغر من $i.low = 11$، لذا سنتجه يمينًا. (لاحظ أنه لا يوجد أي مجال في الشجرة الفرعية اليسرى يتراكب مع i.) إن المجال $[15, 23]$ لا يتراكب مع i، وابنه الأيسر هو $T.nil$، لذا نتجه يمينًا من جديد وتنتهي الحلقة، ويعيد الإجراءُ الحارسَ $T.nil$.

لكي نتمكن من معرفة السبب الذي يجعل INTERVAL-SEARCH صحيحًا، يجب أن نعرف لماذا يكفي أن نستقصي مسارًا وحيدًا منطلقًا من الجذر. تكمن الفكرة الأساسية في أنه مهما كانت العقدة x، إذا كان $x.int$ لا يتراكب مع i، فإن البحث يتجه دومًا باتجاهٍ سليم: فإذا كان ثمة مجال متراكب في الشجرة فلا بد من أن يعثر عليه البحث. تنص المبرهنة التالية على هذه الخاصية بدقة أكبر.

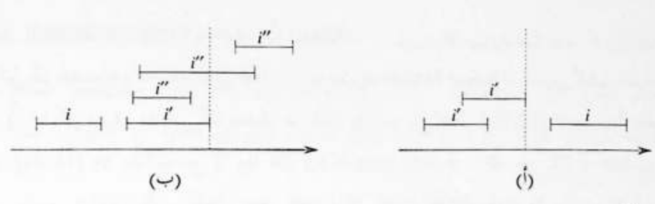
مبرهنة 2.14

إن تنفيذ الإجراء INTERVAL-SEARCH(T, i) إما أن يعيد عقدةً يتراكب مجالها مع i، وإما أن يعيد $T.nil$ وعندئذٍ لا تحتوي الشجرة T على أية عقدة يتراكب مجالها مع i .

البرهان تنتهي الحلقة **while** الموجودة في الأسطر 2-5 عندما $x = T.nil$، أو عندما يتراكب i مع $x.int$. وفي هذه الحالة الأخيرة، من المؤكد أن إعادة x هي نتيجة صحيحة. لذا، سنركز على الحالة السابقة التي تنتهي فيها الحلقة لأن $x = T.nil$.

نستخدم اللامتغير التالي لحلقة **while** في الأسطر 2-5:

إذا كانت الشجرة T تحتوي مجالاً يتراكب مع i، عندها تحتوي الشجرة الفرعية التي جذرها x مثل هذا المجال.



الشكل 5.14 المجالات في إثبات المبرهنة 2.14. تظهر قيمة $x.left.max$ في كل حالة على شكل خط منقط. (أ) يتجه البحث يمينًا. لا يوجد أي مجال i' في الشجرة الفرعية اليسرى لـ x يمكن أن يتراكب مع i. **(ب)** يتجه البحث يسارًا. تحتوي الشجرة الفرعية اليسرى لـ x مجالاً يتراكب مع i (هذه الحالة غير ظاهرة على الشكل)، أو يوجد مجال i' في الشجرة الفرعية اليسرى لـ x بحيث $i'.high = x.left.max$. لا يزال i لا يتراكب مع i'، ولا يتراكب مع أي مجال i'' في الشجرة الفرعية اليمنى لـ x، لأن $i'.low \le i''.low$.

نستخدم لامتغير الحلقة هذا كما يلي:

الاستبداء: قبل التكرار الأول، يجعل السطرُ 1 العقدةَ x جذرًا لـ T بحيث يتحقق اللامتغير.

المحافظة: ينفِّذ كلُّ تكرار للحلقة **while** أحدَ السطرين 4 أو 5. وسنرى أنَّ لامتغير الحلقة يبقى صحيحًا في كلتا الحالتين.

إذا نُفِّذ السطر 5، وبناء على ما جاء في الشرط في السطر 3، يكون لدينا $x.left = T.nil$ أو $x.left.max < i.low$. فإذا كان $x.left = T.nil$ فمن الواضح أن الشجرة الفرعية التي جذرها $x.left$ لا تحتوي على أي مجال يتراكب مع i، وهكذا فإن وضع $x.right$ في x يحافظ على اللامتغير. لذا، سنفترض أن $x.left \neq T.nil$ وأن $x.left.max < i.low$. وكما يبين الشكل 5.14(أ) فإنه لدينا في كل مجال i' في الشجرة الفرعية اليسرى لـ x

$$\begin{aligned} i'.high &\le x.left.max \\ &< i.low \ . \end{aligned}$$

واعتمادًا على خاصية التفرُّع الثلاثي للمجالات، فإن i' و i لا يتراكبان، ومن ثم، فإن الشجرة الفرعية اليسرى لـ x لا تحتوي على أي مجال يتراكب مع i، وهكذا، فإن وضع $x.right$ في x يحافظ على اللامتغير.

من جهة أخرى، إذا نُفِّذ السطر 4، فإننا سنرى أن المكافئ العكسيَّ للامتغير الحلقة يتحقَّق. بمعنى أنه إذا لم توجد مجالات تتراكب مع i في الشجرة الفرعية التي جذرها $x.left$، فليس هناك مجالٌ في أيِّ مكان في الشجرة يتراكب مع i. وحيث إن السطر 4 قد نُفِّذ، بسبب تَحقُّق الشرط في السطر 3، فإن $x.left.max \ge i.low$. وإضافةً إلى ذلك، وبسبب تعريف الواصفة max، يجب أن يكون هناك

مجال ما i' في الشجرة الفرعية اليسرى لـ x بحيث

$$i'.high = x.left.max$$
$$\geq i.low \,.$$

(يوضح الشكل 5.14(ب) هذه الحالة). وإذ إن i و i' لا يتراكبان، و $i'.high < i.low$ غير محقَّق، فهذا يستتبع، بحسب خاصية التفرُّع الثلاثي للمجالات، أن $i.high < i'.low$. وتحمل عقدُ أشجار المجال مفاتيحَ توافق النقاط الطرفية الدنيا للمجالات، ومن ثم، تقتضي خاصية شجرة البحث أنه مهما يكن المجال i'' في الشجرة الفرعية اليمنى لـ x، يكون لدينا

$$i.high < i'.low$$
$$\leq i''.low \,.$$

حسب خاصية التفرُّع الثلاثي للمجالات، لا يتراكب i و i''. ونستنتج أنه سواء وُجد مجال في الشجرة الفرعية اليسرى لـ x يتراكب مع i أم لم يوجد، فإن وضع $x.left$ في x يحافظ على اللامتغير.

الانتهاء: عندما تنتهي الحلقة في حالة $x = T.nil$، فهذا يعني أنه لا يوجد مجال يتراكب مع i في الشجرة الفرعية التي جذرها x. يقتضي المكافئ العكسي للامتغير الحلقة أن T لا تحتوي على أي مجال يتراكب مع i. ومن ثم فإن إعادة $x = T.nil$ هي نتيجة سليمة. ■

وهكذا، فإن الإجراء INTERVAL-SEARCH يعمل بطريقة سليمة.

تمارين

1-3.14

اكتب شبه الرماز لـ LEFT-ROTATE الذي يعمل على العقد في شجرة مجالات ويُحدِّث الواصفات max في زمن $O(1)$.

2-3.14

أعد كتابة رماز INTERVAL-SEARCH بحيث يعمل بطريقة صحيحة عندما تكون جميع المجالات مفتوحة.

3-3.14

صف خوارزميةً فعالةً تعيد، في حالة وجود مجال i معطى، مجالاً يتراكب مع i بحيث تكون لديه أصغر نقطةٍ طرفيةٍ دنيا، أو تعيد $T.nil$ في حال عدم وجود مثل هذا المجال.

4-3.14

لتكن T شجرة مجالات معطاة، والمجال i. صِفْ كيف يمكن سرد جميع المجالات في T التي تتراكب مع i خلال زمن $O(\min(n, k \lg n))$، حيث k عدد المجالات في لائحة الخرج. (*تلميح*: توجد طريقة بسيطة تولِّد عدة استعلامات، وتعدِّل الشجرة بين الاستعلامات. ويوجد حل آخر أكثر تعقيدًا بقليل لا يعدِّل الشجرة.)

5-3.14

اقترح تعديلات على إجراءات شجرة المجالات بحيث تدعم العملية الجديدة INTERVAL-SEARCH-EXACTLY(T, i)، حيث T شجرة مجالات، و i مجال. تعيد العمليةُ مؤشرًا إلى عقدة x في T عندما يكون $x.int.low = i.low$ و $x.int.high = i.high$، أو تعيد $T.nil$ إذا كانت T لا تحوي مثل هذه العقدة. يجب أن تنفَّذ جميع العمليات، ومن ضمنها INTERVAL-SEARCH-EXACTLY في زمن $O(\lg n)$ في شجرة مجالات ذات n عقدة.

6-3.14

بين كيف نحافظ على مجموعة ديناميكية Q من الأعداد التي تدعم العملية MIN-GAP، وهي العملية التي تعطي طويلة الفرق بين أقرب عددين في Q. مثلاً إذا كانت $Q = \{1, 5, 9, 15, 18, 22\}$ عندها تعيد MIN-GAP(Q) القيمة 3 = 15-18 لأن 15 و 18 هما أقرب عددين أحدهما من الآخر في Q. اجعل العمليات INSERT و DELETE و SEARCH و MIN-GAP فعالّة قدر المستطاع وحلّل أزمنة تنفيذها.

★ ***7-3.14***

تمثّل قواعدُ معطيات VLSI عمومًا دارةً متكاملةً على شكل لائحةٍ من المستطيلات. افترض أن كلَّ مستطيلٍ موجَّهٌ بحيث توازي أضلاعُه المحورين x و y، وبحيث نمثّل كلَّ مستطيلٍ بأدنى وأعلى قيمةٍ لإحداثياته على x و y. أعطِ خوارزمية تعمل في زمن $O(n \lg n)$ لتحديد كون مجموعة المستطيلات n الممثَّلة بما ذكرناه آنفًا تحتوي على مستطيلات متراكبة، أو لا. لا تحتاج خوارزميتك إلى إيراد جميع الأزواج المتقاطعة، ولكنها يجب أن تشير إلى وجود تَراكبٍ في حالة غطّى مستطيلٌ مستطيلاً آخر تمامًا، وإن كانت خطوط المحيط لا تتقاطع. (*تلميح*: حرِّك خطًّا "ماسحًا" عبر مجموعة المستطيلات.)

مسائل

1.14 *نقطة التَّراكب الأعظمي*

افترض أننا نرغب في تتبُّع ***نقطة التَّراكب الأعظمي point of maximum overlap*** في مجموعة من المجالات، وهي النقطة التي يتراكب فيها أكبر عدد من مجالات المجموعة.

أ. بيّن أنه يوجد دومًا نقطةَ تَراكبٍ أعظمي هي نقطةٌ طرفيةٌ لإحدى المقتطعات.

ب. صمِّم بنيةَ معطيات تدعم دعمًا فعالاً العمليات: INTERVAL-INSERT و INTERVAL-DELETE و FIND-POM التي تعيد نقطة التَّراكب الأعظمي. (*تلميح*: احتفظ بشجرة حمراء–سوداء لجميع النقاط الطرفية، وأسند القيمة 1+ لكل نقطة طرفية يسرى، والقيمة 1- لكل نقطة طرفية يمنى. أغنِ كل عقدة في الشجرة بالمعلومات الإضافية اللازمة للمحافظة على نقطة التراكب الأعظمي.)

2.14 تبديل جوزفوس

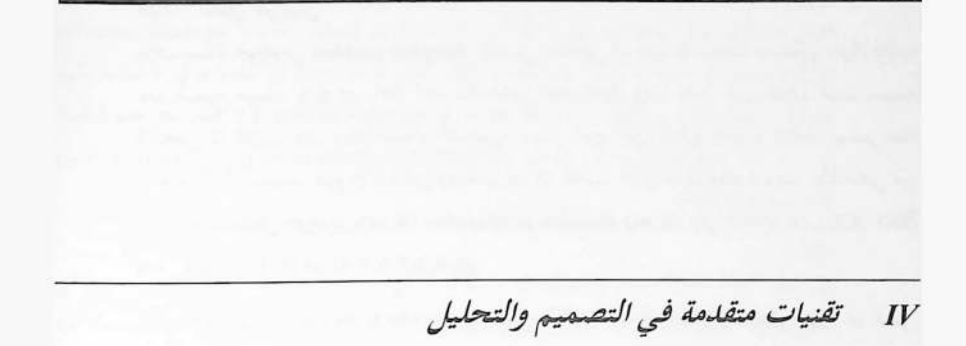
تعرَّف ***مسألة جوزفوس Josephus problem*** كما يلي: لنفترض أن لدينا n شخصًا مصطفِّين دائريًّا، ولدينا عدد صحيح موجب $m \leq n$. نختار أحد الأشخاص نقطة بداية، ونبدأ بالعدّ حول الدائرة بحيث نستبعد الشخص ذا الترتيب m. وبعد استبعاد الشخص، يستمر العد حول الدائرة الجديدة الناتجة. يستمر هذا الإجراء إلى أن يُستبعد جميع الأشخاص وعددهم n. إن الترتيب الذي جرى وفقه استبعاد الأشخاص من الدائرة يعرِّف ***تبديل جوزفوس-(n,m)-Josephus permutation (n,m)*** على الأعداد $1,2,\ldots,n$. فمثلاً *تبديل جوزفوس*-$(7,3)$ هو $(3,6,2,7,5,1,4)$.

أ. افترض أن m ثابت، وأن n عددٌ صحيحٌ معطى. صِفْ خوارزميةً زمنها $O(n)$ تعطي خرجًا هو تبديل جوزفوس $-(n,m)$.

ب. افترض أن m ليس ثابتًا، وأن n و m عددان صحيحان. صِفْ خوارزميةً زمنها $O(n\lg n)$ تعطي خرجًا هو تبديل جوزفوس $-(n,m)$.

ملاحظات الفصل

يصف Preparata و Shamos [282] في كتابهما عددًا من أشجار المجالات التي تظهر في الأدبيات المرجعية، مستشهدَيْن بأعمال H. Edelsbrunner (1980) و E. M. McCreight (1981). يعرض الكتابُ شجرةَ مجالات تتيح لنا، إذا أُعطينا قاعدةَ معطيات ساكنة ذات n مجالاً، أن نعدِّد المجالات k التي تتراكب مع استعلامٍ معطى خلال زمن $O(k+\lg n)$.



IV تقنيات متقدمة في التصميم والتحليل

تمهيد

يدرس هذا الباب ثلاث تقنيات تُستعمَل في تصميم الخوارزميات الفعالة وتحليلها: البرمجة الديناميكية (الفصل 15)، والخوارزميات الشرهة (الفصل 16)، والتحليل المخمَّد (الفصل 17). وقد عَرَضت الأجزاءُ السابقة تقنياتٍ أخرى واسعةَ التطبيق، مثل تقنية فرِّق-تَسُد divide and conquer، وتقنيات إضافة العشوائية وكيفية حل التكرارات. على أن التقنيات الواردةَ في هذا الباب أكثرُ تطورًا وتعقيدًا إلى حدٍّ ما، غير أنها مفيدةٌ لنا في معالجة الكثير من المسائل الحسابية، علمًا بأن الأفكار الأساسية لموضوعات هذا الباب ستتكرَّر لاحقًا في الكتاب.

تُطبَّق البرمجة الديناميكية نموذجيًّا في مسائل الأمثَلة التي تتطلب أخْذَ مجموعةٍ من الخيارات للوصول إلى الحل الأمثل. ومع كلِّ خيارٍ يُتَّخذ، كثيرًا ما تنشأ مسائل جزئية (ثانوية) لها طابع المسائل الأصلية نفسُه. وتكون البرمجة الديناميكية فعالة حين يمكن أن تنشأ مسألةٌ جزئية من أكثر من مجموعةٍ جزئيةٍ واحدة من الخيارات؛ وتتمثَّل التقنية الرئيسيةُ في خزن حلول جميع المسائل الجزئية هذه، فلربما عادت إلى الظهور ثانية. ويبيِّن الفصل 15 كيف تستطيعُ هذه الفكرةُ السهلة، أحيانًا، أن تحوِّل خوارزمياتٍ ذات زمن أسي إلى خوارزميات ذات زمن كثير حدودي.

تُطبَّق الخوارزمياتُ الشرهة، شأنَ خوارزميات البرمجة الديناميكية على مسائل الأمثَلَة التي تتطلب تحديد مجموعةٍ من الخيارات للوصول إلى الحل الأمثل. تكمن فكرة الخوارزمية الشرهة في تحديد كل خيار بطريقة أمثلية محليًّا. ومن الأمثلة البسيطة على هذا "صرف القطع النقدية": فلِكَيْ نجعل عددَ القطع النقدية الأمريكية اللازمة لصرف مبلغٍ معيَّن من المال أصغريًّا؛ يكفي أن نكرِّر اختيار أكبر فئة قطعة نقدية بحيث لا تتجاوز المبلغ المتبقي. ويتيح النَّهْجُ الشره حلاً أمثليًّا لمسائل كثيرةٍ كهذه بسرعةٍ أكبر بكثيرٍ مما قد يتيحه نَهْجُ البرمجة الديناميكية. ومع ذلك، فليس من السهل دومًا الحكم على أن النهجَ الشره سيكون فعالاً أم لا. ويعرِّف الفصل 16 بنظرية الكيانات المصفوفية matroid theory، التي توفِّر أساسًا رياضيًّا يساعدنا على إثبات أن الخوارزمية الشرهة تعطي حلاً أمثلَ.

ونستخدم التحليلَ المخمد لتحليل خوارزمياتٍ معيَّنة تنجز متتالية من العمليات المتشابهة. وعوضًا عن الحد من كلفة متتالية من العمليات بالحد من الكلفة الفعلية لكل عملية منفردة، يمكن استخدام التحليل المخمد لتزويدنا بالحد الأعلى للكلفة الفعلية لكامل المتتالية. ومن مزايا هذا النهج أنه في حين قد تكون بعضُ العمليات مكلفة، فثمة الكثير من العمليات الأخرى التي قد تكون رخيصة. بكلمات أخرى، يمكن أن تُنفَّذ العديد من العمليات في زمن أقل بكثير من زمن التنفيذ في أسوأ الحالات. على أن التحليل المخمد ليس مجرد أداة تحليل، ولكنه أيضًا طريقة للتفكير في تصميم الخوارزميات، بالنظر إلى أن تصميمَ خوارزميةٍ وتحليلَ زمن تنفيذها عمليتان مترابطتان غالبًا إلى حدٍّ بعيد. يقدِّم الفصل 17 ثلاث طرق لتنجيز التحليل المخمد لخوارزمية ما.

15 البرمجة الديناميكية

البرمجة الديناميكية، شأنَ طريقة "فرِّقْ-تَسُد"، تحلُّ مسائلَ بضمِّ حلولٍ لمسائل جزئية. (تشير "البرمجة" في هذا السياق إلى طريقةٍ مُجدوَلة، وليس إلى كتابة رماز حاسوبي.) وكما رأينا في الفصلين 2 و 4، فإن خوارزميات فرِّق-تسد تجزِّئ المسألة إلى مسائل جزئيةٍ منفصلة (مستقلة)، وتُحُلُّ المسائلَ الجزئية عوديًّا، ثم تضم حلولها بغيةَ حلِّ المسألة الأصلية. بالمقابل، تُطبَّق البرمجة الديناميكية dynamic programming حين لا تكون المسائل الجزئية مستقلة، أي حين تتشارك المسائل الجزئية subproblems في مسائل أكثر جزئية subsubproblems. وفي هذا السياق، فإن خوارزمية "فرِّق-تسد" تقوم بعمل أكثر من المطلوب، وذلك بحل المسائل الأكثر جزئية تكراريًّا (مرة بعد مرة). أما خوارزمية البرمجة الديناميكية فتَحلُّ كلَّ مسألة جزئية (أكثر جزئية) مرة واحدة فقط ثم تحتفظ بالإجابات في جدول، متلافيةً بذلك العمل على إعادة حساب الجواب في كل مرة تَحلُّ فيها كلَّ مسألةٍ جزئية ثانوية.

تُطبَّق البرمجة الديناميكية نموذجيًّا على ***مسائل الأَمْثَلَة optimization problems***. ومثل هذه المسائل قد تنطوي على عدة حلولٍ ممكنة، لكلِّ حلٍّ منها قيمة، ونودُّ إيجاد الحل ذي القيمة المثلى (الصغرى أو العظمى). نسمي مثل هذا الحل حلاًّ أمثلَ *an* optimal solution للمسألة، مقابل ما يسمى الحل الأمثل *the* optimal solution، إذ من الممكن أن توجد عدة حلول تحقِّق هذه القيمة المثلى.

ولدى تطوير خوارزمية برمجة ديناميكية، نتّبع متتاليةً من أربع خطوات:

1. وصِّف البنيان لحلٍّ أمثل.
2. عرِّف تكراريًّا القيمة لحلٍّ أمثل.
3. احسب القيمة لحل أمثل بطريقة صعودية من القعر إلى القمة.
4. أنشئ حلاًّ أمثلَ مستقًى من المعلومات المحسوبة.

تؤلِّف الخطواتُ 1-3 الأساسَ لحل مسألة بالبرمجة الديناميكية. وإذا اقتصرت حاجتُنا على قيمة حلٍّ أمثل، لا على الحل الأمثل نفسه، فيمكننا عندئذٍ حذف الخطوة الرابعة. وعند إنجازنا الخطوةَ الرابعة، نحتفظ أحيانًا

بمعلومات إضافية أثناء حساب الخطوة الثالثة لتسهيل إنشاء حل أمثل.

في المقاطع الآتية تُستخدم طريقةُ البرمجة الديناميكية لحلّ بعض مسائل الأمثَلة؛ فيدرس المقطع 1.15 مسألة تقطيع قضيب إلى قضبان أقصر طولاً بحيث نجعل قيمتها الكلية عظمى. ويطرح المقطع 2.15 كيف يمكن ضرب سلسلة من المصفوفات بينما ننجز أقل عدد كلي من الجداءات السُلمية. وفي ضوء هذه الأمثلة على البرمجة الديناميكية، يناقش المقطع 3.15 خاصتين أساسيتين يجب أن تتمتع بهما مسألة ما، كي تكون تقنية حلها بالبرمجة الديناميكية قابلة للتطبيق. ثم يبين المقطع 4.15 طريقةَ إيجاد أطول متتالية جزئية مشتركة لمتتاليتين. أخيرًا يَستخدم المقطعُ 5.15 البرمجةَ الديناميكية لبناء أشجار بحث ثنائية أمثلية، إذا عُلِمَ توزُّع المفاتيح المنشودة.

1.15 تقطيع القضبان

يستخدم مثالُنا الأول البرمجةَ الديناميكية لحل مسألةٍ بسيطة تتعلّق بتحديد مكان قطْع قضبانٍ فولاذية. تشتري شركة سيرلينك Serling Enterprises قضبانًا فولاذيةً طويلة وتقطعها إلى قضبان أقصر، ثم تبيعها. تجري عمليات القطع بدون تكلفة. وتود إدارةُ شركة Serling معرفة أفضل طريقة لقطع القضبان.

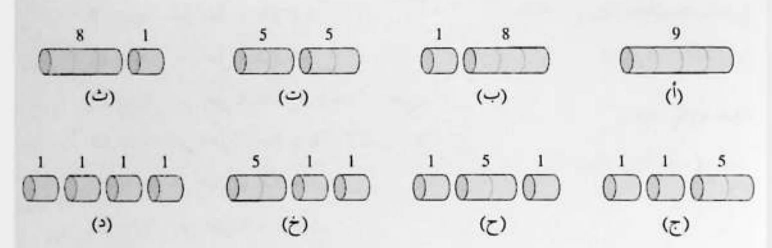
نفترض أننا نعلم القيمة p_i بالدولار، حيث $i = 1,2,\ldots$، التي تطلبها شركة Serling عن كلّ قضيب طوله i إنشًا، علمًا بأن أطوال القضبان هي أعدادٌ صحيحة من الإنشات دومًا. يعطي الشكل 1.15 مثالاً على جدول الأسعار.

إن ***مسألة تقطيع القضبان rod-cutting problem*** هي التالية: لدينا قضيب طوله n إنشًا وجدول بالأسعار p_i، حيث $i = 1,2,\ldots,n$. حدّد الدخل الأعظم r_n الذي يمكن الحصول عليه من تقطيع القضيب وبيع القطع. لاحظ أنه إذا كان السعر p_n لقضيب طوله n كبيرًا بقدر كافٍ، فقد لا يتطلب الحل الأمثل أيَّ عملية قطع على الإطلاق.

لندرس الحالة التي فيها $n = 4$. يبيّن الشكل 2.15 كل الطرق لقطع قضيب طوله 4 إنشات، ومنها الطريقة التي ليس فيها أية عملية قطع على الإطلاق. نرى أن تقطيع القضيب إلى قطعتين، طول كل منهما إنشان، يعطي دخلاً $p_2 + p_2 = 5 + 5 = 10$، وهو دخلٌ أمثلُ.

الطول i	1	2	3	4	5	6	7	8	9	10
السعر p_i	1	5	8	9	10	17	17	20	24	30

الشكل 1.15 مثال على جدول أسعار القضبان. كل قضيب طوله i إنشًا يُكسِب الشركة دخلاً قدره p_i دولارًا.



الشكل 2.15 الطرق الثماني الممكنة لتقطيع قضيب طوله 4. يشير الرقم فوق كل قطعة إلى قيمة تلك القطعة، بحسب جدول مثال الأسعار المبيّن في الشكل 1.15. الاستراتيجية المثلى هي الجزء (ت)، تقطيع القضيب إلى قطعتين طول كل منهما 2، والتي قيمتها الكلية هي 10.

يمكننا تقطيع قضيب طوله n بـ 2^{n-1} طريقة مختلفة، إذ لدينا خيارٌ مستقل للقطع أو عدم القطع، عند المسافة i إنشًا من الطرف الأيسر، حيث $i = 1, 2, \dots, n-1$.[1] سنشير إلى التجزئة إلى قطع باستخدام تدوين الجمع العادي، بحيث تشير العلاقة $7 = 2 + 2 + 3$ إلى تقطيع قضيب طوله 7 إنشات إلى ثلاث قطع؛ اثنتان بطول 2 والثالثة بطول 3. إذا قَطَّع حلٌّ أمثل القضيبَ إلى k قطعة، حيث $1 \le k \le n$، عندها تُقدِّم تجزئةٌ مثلى للقضيب

$$n = i_1 + i_2 + \cdots + i_k$$

إلى قطع أطوالها $i_1, i_2, \dots, i_k$ دخلاً أعظميًا موافقًا قدره:

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k} \ .$$

يمكننا بالاستقصاء، في مثال مسألتنا، تحديد العوائد المثلى r_i في حال $i = 1, 2, \dots, 10$، مع التجزئة المثلى الموافقة

$r_1 = 1$ من الحل $1 = 1$ (لا يوجد أي قطع)،

$r_2 = 5$ من الحل $2 = 2$ (لا يوجد أي قطع)،

$r_3 = 8$ من الحل $3 = 3$ (لا يوجد أي قطع)،

---

[1] إذا أردنا أن تكون الأجزاء المقطَّعة بترتيب القياس غير المتناقص، كان لدينا عدد أقل من طرق التقطيع تؤخذ بالحسبان. ففي حالة $n = 4$، يكون لدينا 5 طرق فقط كهذه: الأجزاء (أ) و (ب) و (ت) و (ث) و (ج) في الشكل 2.15. يسمى عدد الطرق دالة التجزئة partition function؛ وهو يساوي تقريبًا $e^{\pi\sqrt{2n/3}}/4n\sqrt{3}$. وهذه الكمية أقل من 2^{n-1}، ولكنها تبقى أكبر بكثير من أي كثير حدود في n. إلا أننا لن نتابع هذا الخط من النقاش لاحقًا.

$r_4 = 10$ من الحل $4 = 2 + 2$،

$r_5 = 13$ من الحل $5 = 2 + 3$،

$r_6 = 17$ من الحل $6 = 6$ (لا يوجد أي قطع)،

$r_7 = 18$ من الحل $7 = 1 + 6$ أو $7 = 2 + 2 + 3$،

$r_8 = 22$ من الحل $8 = 2 + 6$،

$r_9 = 25$ من الحل $9 = 3 + 6$،

$r_{10} = 30$ من الحل $10 = 10$ (لا يوجد أي قطع).

وبتعميم أكثر، يمكننا استنباط القيم r_n في حال $n \geq 1$ بدلالة الدخول المثلى من قضبان أقصر:

$$r_n = \max\left(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1\right) . \tag{1.15}$$

يوافق المحدِّد الأول p_n، عدم إجراء أي قطع على الإطلاق، وبيع القضيب الذي طوله n كما هو. أما المحدِّدات الـ $n - 1$ الأخرى فتوافق الدخل الأعظم الذي نحصل عليه بإجراء قطع أولي للقضيب إلى قطعتين طولهما i و $n - i$، لكل قيم $i = 1, 2, \ldots, n - 1$. ثم تقطيع هذه القطع تقطيعًا أمثليًا آخر، للحصول على الدخول r_i و r_{n-i} من هاتين القطعتين. ولأننا لا نعلم مقدَّمًا أي قيمة لـ i تجعل الدخل أعظميًّا، علينا دراسة جميع القيم الممكنة لـ i وأخذ القيمة التي تجعل الدخل أعظميًا. لدينا أيضًا خيار عدم أخذ أي قيمة لـ i إذا استطعنا الحصول على دخلٍ أكبر ببيع القضيب دون قطع.

لاحظْ أنه لحل المسألة الأصلية التي حجمها n، نحل مسائل أصغر من النوع نفسه ولكن بحجوم أصغر. وبإجراء القطع الأول، يمكننا اعتبار القطعتين منتسختيْن مستقلين من مسألة تقطيع القضبان. يضم الحل الأمثل الكلي حلَّيْن أمثليْن للمسألتين الجزئيتين المترابطتين، بتعظيم الدخل من كلٍّ من القطعتين. نقول إن مسألة تقطيع القضبان تظهر بنيانًا جزئيًا أمثل ***optimal substructure***: أي إن الحلول المثلى لمسألة تتضمن حلولاً مثلى لمسائل جزئية مرتبطة، يمكن حلها على نحو مستقل.

وبطريقة مرتبطة بترتيب بنيان عودي لمسألة تقطيع القضبان، لكنها أبسط قليلاً، فإننا ننظر إلى التجزئة على أنها تتكون من قطعة أولى طولها i مقطوعة من الطرف الأيسر، ثم بقية من الطرف اليمن طولها $n - i$، علمًا بأن هذه القطعة المتبقية فقط، وليس القطعة الأولى، هي التي يمكن تجزئتها أكثر فأكثر. ويمكننا أن ننظر إلى كل تجزئة لقضيب طوله n بهذه الطريقة: كقطعة أولى متبوعة ببعض التجزئات للقطعة المتبقية. عندما نفعل ذلك، يمكننا صياغة الحل الذي ليس فيه أي قطع على الإطلاق بالقول أن طول القطعة الأولى $i = n$ والدخل p_n وطول القطعة المتبقية 0 والدخل الموافق لها $r_0 = 0$. وبذلك نحصل على النسخة المبسطة التالية للمعادلة (1.15):

$$r_n = \max_{1 \leq i \leq n}\left(p_i + r_{n-i}\right) . \tag{2.15}$$

في هذه الصياغة، يتضمن الحل الأمثل حلاً لمسألة جزئية مرتبطة *واحدة* فقط - وهي القطعة المتبقية- عوضًا عن مسألتين جزئيتين.

تنفيذ نزولي عودي

ينفذ الإجراء التالي الحسابات الضمنية في المعادلة (2.15) بطريقة مباشرة نزولية top-down عودية.

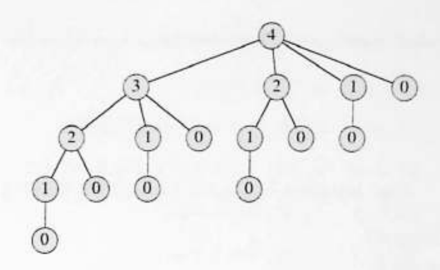
```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = −∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n − i))
6  return q
```

يأخذ الإجراء CUT-ROD كدخل صفيفة $p[1..n]$ للأسعار وعددًا صحيحًا n، ويعيد الدخلَ الأعظم الممكن لقضيب طوله n. فإذا كانت $n = 0$، فلا يوجد أي دخل ممكن، وبذلك يعيد الإجراءُ CUT-ROD القيمةَ 0 في السطر 2. يستبدئ السطر 3 بجعل قيمة الدخل الأعظمي q تساوي $-\infty$، بحيث تَحسب الحلقة **for** في السطرين 4 و 5 القيمةَ الصحيحة لـ $q = \max_{1\le i\le n}(p_i + \text{CUT-ROD}(p, n-i))$؛ ثم يعيد السطر 6 هذه القيمة. وباستقراء بسيط لـ n يتبيَّن أن هذا الجواب يساوي فعلاً الجواب المرغوب فيه r_n باستخدام المعادلة (2.15).

وإذا كان عليك كتابة رماز الإجراء CUT-ROD باستخدام لغة البرمجة المفضلة لديك وتنفيذه على حاسوبك، لوجدت أنه حالما يصبح طول الدخل كبيرًا كبرًا متوسطًا، يستغرق تنفيذُ برنامجك وقتًا طويلاً. فحين تكون $n = 40$، ستجد أن برنامجك يستغرق عدة دقائق على الأقل، وربما استغرق أكثر من ساعة. وستجد عمليًّا أن زمن تنفيذ برنامجك يتضاعف تقريبًا كلما زادت قيمة n بمقدار 1.

ما هو سبب عدم فعالية CUT-ROD؟ تكمن المشكلة في أن CUT-ROD تستدعي نفسها عوديًّا مرات ومرات بنفس قيم الموسطات؛ فهي إذن تحل المسائل الجزئية ذاتها تكراريًّا. يبيّن الشكل 3.15 ما يحدث في حالة $n = 4$: $\text{CUT-ROD}(p, n)$ تستدعي $\text{CUT-ROD}(p, n-i)$ لقيم $i = 1, 2, \ldots, n$. وهذا يكافئ استدعاء $\text{CUT-ROD}(p, n)$ لـ $\text{CUT-ROD}(p, j)$ لقيم $j = 0, 1, \ldots, n-1$. وحين نفك طي الإجرائية عوديًّا، فإن حجم العمل يزداد ازديادًا انفجاريًّا بدلالة n.

ولتحليل زمن تنفيذ CUT-ROD، لِنُشر بـ $T(n)$ إلى عدد الاستدعاءات الكلي للإجرائية CUT-ROD حين يكون الموسط الثاني له مساويًا n عند استدعائه. وهذا التعبير يساوي عددَ العقد في الشجرة الفرعية التي لصيقة جذرها n في الشجرة العَوْدية. يتضمن العدّ الاستدعاء الأولي عند الجذر. وبذلك



الشكل 3.15 الشجرة العَوْدية التي تبيّن الاستدعاءات العودية الناتجة عن استدعاء $\text{CUT-ROD}(p,n)$ في حالة $n = 4$. تعطي كل لصيقة عقدة الحجمَ n للمسألة الجزئية الموافقة، وبذلك توافق الوصلة من أبٍ لصيقتُه s إلى ابنٍ لصيقتُه t تقطيعَ قطعة أولية حجمها $s - t$ وترك مسألةٍ جزئيةٍ متبقية حجمها t. المسار من الجذر إلى ورقة يوافق إحدى الطرق الـ 2^{n-1} لقطع قضيب طوله n. وعمومًا، يكون للشجرة العودية هذه 2^n عقدة و 2^{n-1} ورقة.

يكون، $T(0) = 1$ و

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) \ . \tag{3.15}$$

الرقم 1 هو للاستدعاء عند الجذر، والحد $T(j)$ يحصي عدد الاستدعاءات (ومنها الاستدعاءات العودية) الناجمة عن الاستدعاء $\text{CUT-ROD}(p, n - i)$، حيث $j = n - i$. يُطلب إليك في التمرين 15.1-1 أن تبين أن:

$$T(n) = 2^n \ , \tag{4.15}$$

وبذلك يكون زمن تنفيذ CUT-ROD أسيًّا في n.

وبإعادة النظر فيما قد سلف نجد أن زمن التنفيذ الأُسِّي هذا ليس مفاجئًا كثيرًا؛ فمن الواضح أن CUT-ROD تأخذ بالحسبان كل الطرق 2^{n-1} الممكنة لقطع قضيب طوله n. ويُذكَر أن لشجرة الاستدعاءات العودية 2^{n-1} ورقة، واحدة لكل طريقة ممكنة لقطع القضيب. تعطي اللصيقات على المسار البسيط من الجذر إلى إحدى الأوراق حجومَ كل القطع المتبقية اليمنى قبل كل عملية قطع؛ أي إن اللصيقات تعطي نقاط القطع الموافقة، مَقِيسةً من الطرف الأيمن للقضيب.

استخدام البرمجة الديناميكية لقطع القضبان الأمثل

سنبيّن الآن كيف نحوّل CUT-ROD إلى خوارزمية فعالة باستخدام البرمجة الديناميكية.

تعمل طريقة البرمجة الديناميكية كما يلي. بعد ملاحظة عدم فعالية الحل العودي البسيط لأنه يحل المسائل الجزئية نفسها تكراريًّا، نتخذ الترتيبات اللازمة لحل كل مسألة جزئية *مرة واحدة* فقط، وتخزين حلّها. فإذا لَزِمَنا العودة إلى حل هذه المسألة الجزئية ثانيةً لاحقًا، يكفي أن نتفقَّدها دون الحاجة إلى إعادة حسابها. وبذلك

تَستخدم البرمجةُ الديناميكية ذاكرةً إضافية لاختصار زمن الحساب؛ وهي بذلك تقدم مثالاً على ***المقايضة بين الزمن والذاكرة time-memory trade-off***. ويمكن أن يكون الكسب الزمني هائلاً: إذ يمكن تحويل الحل بزمنٍ أُسّي إلى حلٍّ بزمنٍ كثير حدودي. ويُنفَّذ أسلوبُ البرمجة الديناميكية بزمن كثير حدودي عندما يكون عدد المسائل الجزئية *المتمايزة* ذات الصلة كثير حدودي في حجم المداخل، ويمكننا حل كل مسألة جزئية منها بزمن كثير حدودي.

يوجد عادة طريقتان متكافئتان لتنجيز أسلوب البرمجة الديناميكية، سنوضحهما في مثالنا المتعلِّق بقطع القضبان.

الأسلوب الأول ***نزولي مع استذكار top-down with memoization***[2]، وبمقتضاه نكتب الإجراء عوديًّا بطريقة طبيعية، ولكن نُعدِّله بحيث نخزن نتيجة كل مسألة جزئية (عادة في صفيفة أو جدول تلبيد). الإجراء الآن يتحقَّق أولاً ليرى إن كانت المسألة الجزئية محلولة سابقًا. فإذا كان الأمر كذلك، أعادَ القيمةَ المخزنة، مختصرًا مزيدًا من الحسابات عند هذا المستوى؛ وإلاّ، فإنه يحسب القيمة بالطريقة المعتادة. ونقول إن الإجراء العودي قد جرى ***استذكاره memoized***؛ فهو "يتذكر" النتائج التي حسبها سابقًا.

أما الأسلوب الثاني فهو ***الطريقة الصعودية bottom-up method***. يعتمد هذا الأسلوب عادةً على مفهومٍ طبيعي ما لـ "حجم" المسألة الجزئية، بحيث يَعتمد حلُّ مسألةٍ جزئية معيَّنة اعتمادًا كاملاً على حلّ مسائل جزئية "أصغر". نفرز المسائل الجزئية وفقًا لحجمها ونحلها بحسب ترتيب حجمها بحيث نبدأ بأصغرها. فإذا أتينا إلى حلّ مسألةٍ جزئية معينة، نكون قد حللنا قبلها كل المسائل الجزئية التي هي أصغر منها، والتي يعتمد عليها حل هذه المسألة الجزئية، وخزّنا تلك الحلول. نحلُّ كلَّ مسألة جزئية مرةً واحدةً فقط، وعندما نراها أول مرة نكون قد حللنا كل المسائل الجزئية التي تتطلبها.

هذان الأسلوبان يعطيان خوارزميتين لهما زمن التنفيذ المقارب نفسه، باستثناء الظروف الاستثنائية، حيث يمتنع الأسلوب التُّزولي عن العمل عوديًّا لسبر جميع المسائل الجزئية الممكنة. أما الأسلوب الصعودي فيتمتع غالبًا بعوامل ثابتة أفضل، لأن حملها الإضافي من استدعاءات الإجراء أقل.

نورد فيما يلي شبه الرماز للإجراء النُّزولي CUT-ROD مع إضافة الاستذكار:

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = −∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

[2] ليس هذا خطأً في رسم الكلمة؛ فهي *memoization* فعلاً، وليس *memorization*. وهي مستمَدة من memo، بالنظر إلى أن التقنية تتمثل في تسجيل قيمةٍ يمكننا العودة إليها لاحقًا.

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = −∞
6      for i = 1 to n
7          q = max(q, p[i] +MEMOIZED-CUT-ROD-AUX(p, n − i, r))
8  r[n] = q
9  return q
```

هنا، يستبدئ الإجراءُ الرئيسي MEMOIZED-CUT-ROD صفيفةً مساعدة جديدة $r[0..n]$ قيمها تساوي $-\infty$، وهو خيار مناسب نشير به إلى "مجهول". (قيم الدخل revenue المعروفة غير سالبة دومًا.) ثم يستدعي مساقه المساعد MEMOIZED-CUT-ROD-AUX.

الإجراء MEMOIZED-CUT-ROD-AUX هو النسخة مع الاستذكار لإجرائنا السابق CUT-ROD. إنه يدقِّق أولاً في السطر 1 إذا كانت القيمة المرغوبة معروفة سابقًا، فإذا كان الأمر كذلك، يعيد هذه القيمة في السطر 2. وإلا فإنه يحسب في الأسطر 3-7 القيمة المرغوبة q بالطريقة المعتادة، السطر 8 يخزن القيمة في $r[n]$، والسطر 9 يعيدها.

النسخة الصعودية فهي أبسط:

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = −∞
5      for i = 1 to j
6          q = max(q, p[i] + r [j − i])
7      r[j] = q
8  return r[n]
```

في حالة الأسلوب الصعودي للبرمجة الديناميكية، يَستخدم الإجراء BOTTOM-UP-CUT-ROD الترتيب الطبيعي للمسائل الجزئية: فمسألةٌ جزئية حجمُها i "أصغر" من مسألة جزئية حجمها j إذا كان $i < j$. وبذلك يحل الإجراءُ المسائلَ الجزئية التي حجومها $j = 0, 1, \dots, n$، في هذا الترتيب.

السطر 1 في الإجراء BOTTOM-UP-CUT-ROD ينشئ صفيفةً جديدة $r[0..n]$ يخزن فيها نتائج المسائل الجزئية، والسطر 2 يستبدئ $r[0]$ بالقيمة 0 لأن القضيب الذي طوله 0 لا يكسب أي دخل. تحل السطور 3-6 كلَّ مسألةٍ جزئية حجمها j لكلِّ $j = 1, 2, \dots, n$ بترتيب الحجوم المتزايدة. والأسلوب الذي

استُخدم في حل مسألة بحجم معين j هو نفسه المستَخدَم في CUT-ROD، باستثناء أن السطر 6 يعنون الآن مباشرة عنصر الصفيفة $r[j-i]$ بدلاً من القيام باستدعاء عَوْدي لحل المسألة الجزئية التي حجمها $j-i$. يخزن السطر 7 في $r[j]$ حلَّ المسألة الجزئية التي حجمها j. أخيرًا يعيد السطر 8 القيمة $r[n]$ التي تساوي القيمة المثلى r_n.

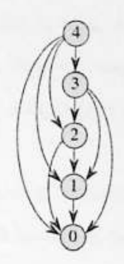
للنسختين الصعودية والنُّزولية زمن التنفيذ المقارب نفسه. زمن تنفيذ الإجراء BOTTOM-UP-CUT-ROD هو $\Theta(n^2)$ بسبب بنية الحلقتين المتداخلتين. يشكل عدد التكرارات في حلقة **for** الداخلية، في السطور 5–6 سلسلةً حسابية. زمن تنفيذ نظيره النُّزولي MEMOIZED-CUT-ROD هو أيضًا $\Theta(n^2)$، مع أن رؤية زمن التنفيذ هذا أصعب بقليل. ولما كان الاستدعاءُ العَوْدي لحل مسألةٍ جزئية جرى حلها سابقًا يعيد القيمة فورًا، فإن MEMOIZED-CUT-ROD يحلُّ كلَّ مسألة جزئية مرة واحدة فقط. إنه يحل المسائل الجزئية ذات الحجوم $0, 1, \ldots, n$. ولحلِّ مسألةٍ جزئية حجمها n، فإن حلقة **for** في السطرين 6–7 تتكرر n مرة. وبذلك يشكل عددُ التكرارات الكلي لحلقة **for** هذه، لكل الاستدعاءات العودية لـ MEMOIZED-CUT-ROD سلسلةً حسابية، تعطي تكراراتٍ عددها $\Theta(n^2)$، تمامًا مثل حلقة **for** الداخلية في BOTTOM-UP-CUT-ROD. (إننا فعليًا نستخدم نوعًا من التحليل المجمَّع هنا، وسنعالج التحليل المجمَّع لاحقًا في المقطع 1.17.)

بيانات المسائل الجزئية

حين نفكر في مسألة برمجة ديناميكية، علينا أن ندرك مجموعةَ المسائل الجزئية ذات الصلة، وكيف تعتمد هذه المسائل الجزئية بعضها على بعض.

يضم ***بيانُ المسألة الجزئية* subproblem graph** للمسألة هذه المعلومات تمامًا. يوضِّح الشكل 4.15 بيان المسألة الجزئية لمسألة تقطيع القضبان في حالة $n = 4$. إنه بيان موجَّه، يتضمن عقدة واحدة لكل مسألة جزئية متمايزة. ولبيان المسألة الجزئية وصلةٌ موجَّهة من عقدة المسألة الجزئية x إلى عقدة المسألة الجزئية y إذا كان تحديد حل أمثل للمسألة الجزئية x يتطلب الأخذ بالحسبان حلاً أمثلَ للمسألة الجزئية y. على سبيل المثال، يتضمن بيان المسألة الجزئية وصلة من x إلى y إذا كان الإجراءُ العَوْدي النُّزولي لحل x يستدعي نفسَه مباشرةً لحل y. يمكننا النظر إلى بيان المسألة الجزئية على أنه نسخة "مختصرة" أو "مدمجة" للشجرة العودية في الطريقة العودية النُّزولية، التي ندمج فيها كل العقد المتعلقة بالمسألة الجزئية نفسها في عقدة واحدة، ونوجّه كل الوصلات من الأب إلى الابن.

تَعتبر الطريقة الصعودية للبرمجة الديناميكية عُقد بيان المسألة الجزئية مرتبةً بحيث نحل المسائل الجزئية y المجاورة لمسألة جزئية معلومة x قبل حل المسألة الجزئية x. (تذكر من المقطع ب.4 أن علاقة التجاور ليست متناظرة بالضرورة.) وباستخدام مصطلحات الفصل 22، لخوارزمية البرمجة الديناميكية الصعودية، فإننا نعتبر العقد في بيان المسألة الجزئية مرتبة حسب "فرز طبولوجي معكوس" أو "فرز طبولوجي للمنقول"



الشكل 4.15 بيان المسألة الجزئية لمسألة تقطيع القضبان في حالة $n = 4$. تعطي لصيقات العقد حجوم المسائل الجزئية الموافقة. تشير الوصلة الموجهة (x,y) إلى أننا نحتاج إلى حل المسألة الجزئية y حين نحل المسألة الجزئية x. يشكل هذا البيان نسخة مختصرة لشجرة البيان في الشكل 3.15 ، التي تندمج فيها جميعُ العقد التي لها اللصيقة نفسها في عقدة واحدة، وتنطلق جميع الوصلات من الأب إلى الابن.

(انظر المقطع 4.22) لبيان المسألة الجزئية. وبعبارةٍ أخرى، لا ندرس أي مسألة جزئية حتى نحل كل المسائل الجزئية التي تعتمد عليها. وبالمثل، وباستخدام مفاهيم من الفصل نفسه، يمكننا النظر إلى الطريقة التُّزولية (مع استذكار) للبرمجة الديناميكية على أنها "بحث في العمق أولاً" لبيان المسألة الجزئية (انظر المقطع 3.22).

يمكن أن يساعدنا حجمُ بيان المسألة الجزئية $G = (V, E)$ على تحديد زمن تنفيذ خوارزمية البرمجة الديناميكية. ولأننا نحل كلَّ مسألة جزئية مرة واحدة فقط، فإن زمن التنفيذ هو مجموع الأزمنة اللازمة لحل كل مسألة جزئية. ويكون زمنُ حساب حل مسألة جزئية متناسبًا عادةً مع درجة العقدة الموافقة في بيان المسألة الجزئية (عدد الوصلات الخارجة منها)، وعددُ المسائل الجزئية مساويًا عددَ العقد في بيان المسألة الجزئية. في هذه الحالة العامة، يكون زمن تنفيذ البرمجة الديناميكية خطيًّا من جهة عدد العقد والوصلات.

إعادةُ بناء الحل

تعيد حلولُنا لمسألة تقطيع القضبان باستخدام البرمجة الديناميكية قيمةَ الحل الأمثل، إلا أنها لا تعيد حلاً فعليًا: لائحة بأطوال القطع. يمكننا توسيع أسلوب البرمجة الديناميكية ليسجل ليس فقط *القيمة* المثلى المحسوبة لكل مسألة جزئية، وإنما أيضًا *الخيار* الذي أفضى إلى القيمة المثلى. بهذه المعلومات، يمكننا الآن طباعة حل أمثل.

وفيما يلي نسخة موسعة من BOTTOM-UP-CUT-ROD تحسب، لكل قضيب طوله j، ليس فقط الدخل الأعظم r_j، بل الطول الأمثل s_j أيضًا لأول قطعة يجب قطعها:

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] and s[0..n] be new arrays
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          if q < p[i] + r[j - i]
7              q = p[i] + r [j - i]
8              s[j] = i
9      r[j] = q
10 return r and s
```

هذا الإجراء مشابه لـ BOTTOM-UP-CUT-ROD، باستثناء أنه ينشئ الصفيفة s في السطر 1، ويُحدِّث $s[j]$ في السطر 8 ليحتفظ بالطول الأمثل i لأول قطعة نقطعها عند حلِّ المسألة الجزئية التي حجمها j.

يأخذ الإجراء التالي لائحة الأسعار p وطول القضيب n، ويستدعي -EXTENDED-BOTTOM-UP CUT-ROD لحساب الصفيفة $s[1..n]$ التي تمثل أطوال القطع الأولى المثلى، ثم يطبع اللائحة الكاملة بأطوال القطع في تقسيمٍ أمثل لقضيب طوله n:

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

في مثالنا لقطع القضبان، فإن الاستدعاء EXTENDED-BOTTOM-UP-CUT-ROD$(p, 10)$ سيعيد الصفيفتين التاليتين:

10	9	8	7	6	5	4	3	2	1	0	i
30	25	22	18	17	13	10	8	5	1	0	$r[i]$
10	3	2	1	6	2	2	3	2	1	0	$s[i]$

إن استدعاء PRINT-CUT-ROD-SOLUTION$(p, 10)$ سيطبع 10 فقط، إلا أن استدعاءها مع $n = 7$ سيطبع القطعتَيْن 1 و 6، الموافقين لأول تقسيم أمثل لـ r_7 المعطى سابقًا.

تمارين

1-1.15

بيّن أن المعادلة (4.15) تَنتج من المعادلة (3.15) والحالة الابتدائية $T(0) = 1$.

2-1.15

بيّن، بإيراد أمثلةٍ معاكسة، أن الاستراتيجية "الشرهة" التالية لا تحدِّد دومًا طريقة مثلى لتقطيع القضبان. عرّف

كثافة density قضيب طوله i على أنها p_i/i، أي قيمته لكل بوصة. تُقطِّع الاستراتيجية الشرهة من قضيب طوله n قطعة أولى طولها i، بحيث $1 \le i \le n$، ذات كثافة عظمى. ثم تستمر بتطبيق الاستراتيجية الشرهة على القطعة المتبقية التي طولها $n - i$.

3-1.15

ادرس تعديلاً على مسألة تقطيع القضبان التي فيها، إضافة إلى سعر كل قضيب p_i، كلفة ثابتة c لكل عملية قطع. إن الدخل المرتبط بكل حلٍّ هو الآن مجموع أسعار القطع مطروحًا منه تكاليف التقطيع. أعطِ خوارزمية برمجة ديناميكية تحل هذه المسألة المعدَّلة.

4-1.15

عدّل الإجراء MEMOIZED-CUT-ROD ليعيد ليس فقط القيمة وإنما الحل الفعلي أيضًا.

5-1.15

تُعرَّف أعداد فيبوناتشي Fibonacci عوديًّا بالعلاقة (22.3). أعطِ خوارزمية برمجة ديناميكية تُنفَّذ بزمن $O(n)$ لحساب عدد فيبوناتشي من المرتبة n. ارسم بيان المسألة الجزئية. ما هو عدد العقد والوصلات في هذا البيان؟

2.15 جداء سلسلة من المصفوفات

مثالنا التالي على البرمجة الديناميكية هو خوارزمية حلّ مسألة جداء سلسلة من المصفوفات. ليكن لدينا متتالية (سلسلة) من n مصفوفة $\langle A_1, A_2, \dots, A_n \rangle$ وعلينا إيجاد جدائها، ونريد أن نحسب الجداء

$$A_1 A_2 \cdots A_n \quad (5.15)$$

يمكننا تقويم (حساب) العبارة (5.15) باستخدام الخوارزمية المعيارية لضرب أزواج المصفوفات باعتبارها مساقًا فرعيًّا، بعد أن نكون قد وضعناها ضمن أقواس لحل كل الجوانب الغامضة المتعلقة بكيفية ضرب المصفوفات بعضها في بعض. ولما كان ضرب المصفوفات عمليةً تجميعية، فإن كل عمليات وضع الأقواس تؤدي إلى الجداء نفسه. نقول عن جداء مصفوفات أنه ***كامل الأقواس fully parenthesized*** إذا تَكَوَّن من مصفوفة وحيدة أو من جداء مصفوفتين كاملتَي الأقواس، محدودتين بأقواس. على سبيل المثال، إذا كانت سلسلة المصفوفات هي $\langle A_1, A_2, A_3, A_4 \rangle$، فإن بإمكاننا أن نجعل الجداء $A_1A_2A_3A_4$ كاملَ الأقواس بخمس طرق متمايزة هي:

$$(A_1(A_2(A_3A_4))) ,$$
$$(A_1((A_2A_3)A_4)) ,$$
$$((A_1A_2)(A_3A_4)) ,$$
$$((A_1(A_2A_3))A_4) ,$$
$$(((A_1A_2)A_3)A_4) .$$

وقد يكون لطريقة وضع الأقواس على سلسلة المصفوفات تأثيرٌ لافتٌ في تكلفة حساب جدائها. لندرس أولاً تكلفة جداء مصفوفتين. تُعطى الخوارزمية المعيارية من خلال شبه الرماز التالي، الذي يعمم الإجراء SQUARE-MATRX-MULTIPLY من المقطع 2.4، علمًا أن الواصفات *rows* و *columns* هي عدد السطور وعدد الأعمدة في مصفوفة.

```
MATRIX-MULTIPLY(A, B)
1  if A.columns ≠ B.rows
2      error "incompatible dimensions"
3  else let C be a new A.rows × B.columns matrix
4      for i = 1 to A.rows
5          for j = 1 to B.columns
6              c_ij = 0
7              for k = 1 to A.columns
8                  c_ij = c_ij + a_ik · b_kj
9      return C
```

ولا نستطيع ضرب مصفوفتين A و B إلا إذا كانتا ***متوافقتين compatible***: إذ لا بدَّ من أن يكون عدد الأعمدة في A مساويًا عدد السطور في B. فإذا كانت A مصفوفة $p \times q$ وكانت B مصفوفة $q \times r$، كانت المصفوفة الناتجة C هي مصفوفة $p \times r$. إن الزمن اللازم لحساب C محكوم بعدد الجداءات السلمية في السطر 8، وهو pqr. نبيِّن فيما يلي التكاليف بدلالة عدد الجداءات السلَّمية.

لبيان التكاليف المختلفة التي يجلبها مختلف أشكال وضع الأقواس لجداء المصفوفات، سندرس مسألة حساب جداء سلسلة ثلاث مصفوفات $\langle A_1, A_2, A_3 \rangle$، ولنفترض أن أبعاد المصفوفات هي 10×100 و 100×5 و 5×50، على التتالي. إذا أجرينا عملية الضرب بحسب الأقواس $((A_1A_2)A_3)$، فإننا ننجز $10 \cdot 100 \cdot 5 = 5000$ عملية جداء سلَّمي لحساب المصفوفة A_1A_2 التي هي مصفوفة 10×5، ثم $10 \cdot 5 \cdot 50 = 2500$ عملية جداء سلَّمي أخرى لحساب ناتج ضرب هذه المصفوفة بالمصفوفة A_3، ويكون المجموع 7500 عملية جداء سلمي. فلو أجرينا، بدلاً من ذلك، عملية الضرب بناءً على وضع الأقواس $(A_1(A_2A_3))$ للَزِمنا $100 \cdot 5 \cdot 50 = 25{,}000$ عملية جداء سلمي لحساب مصفوفة الجداء A_2A_3 مصفوفة 100×5، إضافة إلى $10 \cdot 100 \cdot 50 = 50{,}000$ عملية جداء سلمي لضرب الناتج في المصفوفة A_1، ويكون عدد الجداءات السلمية الكلية هو 75,000. وبذلك يكون حساب الجداء بحسب وضعية الأقواس الأولى أسرع بعشر مرات.

ونصوغ ***مسألة جداء سلسلة من المصفوفات matrix-chain multiplication problem*** كما يلي: إذا كان لدينا سلسلة $\langle A_1, A_2, \ldots, A_n \rangle$ مؤلَّفة من n مصفوفة، حيث $i = 1, 2, \ldots, n$، فإن للمصفوفة A_i الأبعاد $p_{i-1} \times p_i$، فالمسألة هي وضع الأقواس الكاملة للجداء $A_1 A_2 \cdots A_n$ بحيث يكون عدد عمليات

الجداء السلمي أصغريًّا.

لاحِظ أننا في مسألة جداء سلسلة المصفوفات، لا نضرب المصفوفات فعليًّا. وهدفنا فقط تحديد ترتيبٍ لضرب المصفوفات بحيث تكون التكلفة صغرى. عموماً، فإن الوقت المصروف في تحديد الترتيب الأمثل يُعوِّض بأكثر من ثمنه بالزمن المُدَّخر لاحقًا، حين ننجز فعليًّا ضرب المصفوفات (كما في تنفيذ 7500 عملية جداء سلمي فقط عوضًا عن 75,000 عملية).

حساب عدد أوضاع الأقواس

قبل البدء بحل مسألة ضرب سلسلة المصفوفات بواسطة البرمجة الديناميكية، لنُقنِع أنفسنا بأن الاختبار الشامل لكل أوضاع الأقواس الممكنة لا يفضي إلى خوارزمية فعالة. لنعبِّر عن عدد بدائل وضع الأقواس لسلسلة من n مصفوفة بالرمز $P(n)$. فعندما تكون $n = 1$ يكون لدينا مصفوفة واحدة، ومن ثَمّ توجد طريقة واحدة لوضع كامل الأقواس لجداء المصفوفات. وعندما تكون $n \geq 2$ فإن حاصل ضرب جداءَيْن جزئيين كاملَي الأقواس للمصفوفات، ويمكن أن يحدث تفريق الجداءين الجزئيين بين المصفوفتين ذواتَي الترتيب k و $k+1$ لأي قيمة $k = 1, 2, \ldots, n-1$. وبذلك نحصل على التكرار:

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \ , \\ \displaystyle\sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \ . \end{cases} \tag{6.15}$$

وقد طُلب إليك في المسألة 12-4 البرهان على أن الحل لتكرار مشابه هو متتالية من ***أعداد كاتالان Catalan numbers***، التي تتزايد كـ $\Omega(4^n/n^{3/2})$. ثمة تمرين مشابه (انظر التمرين 3-2.15) يطلب البرهان على أن حل التكرار (6.15) هو $\Omega(2^n)$. وبذلك يكون عدد الحلول أسيًّا في n. ولذلك فإن طريقة البحث الشامل تؤدي إلى استراتيجيةٍ رديئة عند تحديد الوضع الأمثل للأقواس في جداء سلسلة مصفوفات.

تطبيق البرمجة الديناميكية

سنستخدم طريقة البرمجة الديناميكية لتحديد الكيفية المثلى لوضع الأقواس لسلسلة مصفوفات. وللقيام بذلك، سنتّبع تتالي الخطوات الأربعة التي ذكرناها في بداية هذا الفصل:

1. وَصِّف البنيان لحلٍّ أمثل.
2. عرِّفْ تكراريًّا قيمةَ حلٍّ أمثل.
3. احسب قيمةَ حلٍّ أمثل.
4. أنشِئ حلاً أمثلَ من المعلومات المحسوبة

وسنتعرَّض لهذه الخطوات بالترتيب، مُبيِّنين بوضوح كيف نطبّق كل خطوة على المسألة.

الخطوة 1: بنية الأقواس المثلى

لتنفيذ خطوتنا الأولى في نموذج البرمجة الديناميكية فإننا نوجد البنية الجزئية المثلى، ثم نستخدمها لبناء حل أمثل للمسألة من الحلول المثلى للمسائل الجزئية. وفي حالة مسألة جداء سلسلة المصفوفات، يمكننا إنجاز هذه الخطوة كما يلي. للتسهيل، سنعتمد التدوين $A_{i..j}$، حيث $i \leq j$ للمصفوفة التي تنتج من حساب الجداء $A_iA_{i+1}\cdots A_j$. لاحظ أنه إذا كانت المسألة غير تافهة، أي $i < j$، وجب أن يفرِّق أيُّ وضعٍ لأقواس الجداء $A_iA_{i+1}\cdots A_j$ هذا الجداءَ بين المصفوفتين A_k و A_{k+1}، حيث k عدد صحيح ما في المجال $i \leq k < j$. أي إننا نحسب أولاً $A_{i..k}$ و $A_{k+1..j}$، لكل قيمة k، ثم نضرب إحداهما في الأخرى لحساب الجداء النهائي $A_{i..j}$. إن تكلفة الحساب بوضع الأقواس هذا هي مجموع تكلفة حساب المصفوفة $A_{i..k}$ وحساب المصفوفة $A_{k+1..j}$ إضافة إلى تكلفة ضرب إحداهما بالأخرى.

البنية الجزئية المثلى لهذه المسألة هي كالتالي. افترض أن الوضع الأمثل للأقواس يفرق الجداء $A_iA_{i+1}\cdots A_j$ بين المصفوفتين A_k و A_{k+1}. عندها يجب أن يكون وضع الأقواس للسلسلة الجزئية "البادئة" $A_iA_{i+1}\cdots A_k$ ضمن الوضع الأمثل للأقواس $A_iA_{i+1}\cdots A_j$ أمثلَ لـ $A_iA_{i+1}\cdots A_k$. لماذا؟ لأنه إذا كان هناك طريقة أقل تكلفة لوضع الأقواس للجداء $A_iA_{i+1}\cdots A_k$، فإن تعويض هذه الأقواس في الوضع الأمثل للأقواس لحساب الجداء $A_iA_{i+1}\cdots A_j$ سينتج وضعًا آخر للأقواس تكلفته أقل من التكلفة المثلى: وهذا تناقض. وتتحقَّق الملاحظة نفسها لوضع الأقواس للسلسلة الجزئية $A_{k+1}A_{k+2}\cdots A_j$ في الوضع الأمثل للأقواس في حساب الجداء $A_iA_{i+1}\cdots A_j$: إذ يجب أن يكون وضع الأقواس أمثلَ لوضع الأقواس $A_{k+1}A_{k+2}\cdots A_j$.

الآن نستخدم بنيتنا الجزئية المثلى لنبيّن أنه يمكننا بناء حل أمثل للمسألة من حلول مثلى للمسائل الجزئية. لقد رأينا أن أيَّ حلٍّ لمتنسَخٍ غير تافه في مسألة جداء سلسلة المصفوفات يتطلب أن نفرق الجداء، وأن أيَّ حلٍّ أمثلَ يتضمن في حدِّ ذاته حلولاً مثلى لمتنسَخاتِ المسائل الجزئية. وبذلك، يمكننا بناء حلٍّ أمثلَ لمتنسَخ جداء سلسلة مصفوفات بتفريق المسألة إلى مسألتَيْن جزئيتَيْن (بوضع أقواس على نحو أمثل لكل من $A_iA_{i+1}\cdots A_k$ و $A_{k+1}A_{k+2}\cdots A_j$)، ثم إيجاد حلول مثلى لمتنسخات المسائل الجزئية، ثم بضم هذه الحلول المثلى للمسائل الجزئية. يجب أن نتأكّد، حين نبحث عن المكان الصحيح لتفريق الجداء، أننا أخذنا كل أماكن التفريق المحتملة بالحسبان، بحيث نتأكد أننا تحرَّينا الوضع الأمثل.

الخطوة 2: حلٌّ عَوْدي

بعد ذلك، نحدّد تكلفة الحل الأمثل عوديًّا بدلالة الحلول المثلى للمسائل الجزئية. وفي حالة مسألة جداء سلسلة

المصفوفات، نعتبر مسائلنا الجزئية هي مسائل تحديد التكلفة الصغرى لوضع الأقواس $A_iA_{i+1}\cdots A_j$ حيث $1 \le i \le j \le n$. ليكن $m[i,j]$ أصغر عدد لعمليات الجداء السلمية اللازمة لحساب المصفوفة $A_{i..j}$؛ عندها، لحل المسألة الكلية، ستكون تكلفة طريقة التكلفة الدنيا لحساب $A_{1..n}$ هي $m[1,n]$.

ويمكننا تحديد $m[i,j]$ عوديًّا كما يلي. إذا كانت $i = j$ فالمسألة تافهة؛ وتتكون السلسلة من مصفوفة واحدة $A_{i..i} = A_i$، وبذلك لا نحتاج إلى أي عمليات جداء سلَّمية لحساب الجداء. وبذلك يكون $m[i,i] = 0$ حيث $i = 1, 2, \ldots, n$. ولحساب $m[i,j]$ عندما تكون $i < j$ نستفيد من بنية الحل الأمثل في الخطوة 1. لنفترض أن وضع الأقواس الأمثل يفرق الجداء $A_iA_{i+1}\cdots A_j$ بين A_k و A_{k+1} حيث $i \le k < j$. حينها يكون $m[i,j]$ مساويًا التكلفة الصغرى لحساب الجداءات الجزئية $A_{i..k}$ و $A_{k+1..j}$، إضافة إلى تكلفة ضرب إحدى هاتين المصفوفتين بالأخرى. لنتذكر أن أبعاد كل مصفوفة A_i هو $p_{i-1} \times p_i$، فنرى أن حساب مصفوفة الجداء $A_{i..k}A_{k+1..j}$ يتطلب إجراء $p_{i-1}p_kp_j$ عملية جداء سلَّمي. وبذلك يصبح لدينا:

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j \ .$$

تفترض هذه المعادلة العودية أننا نعلم قيمة k، ولكن الأمر ليس كذلك. على أن ثمة $j - i$ قيمة ممكنة فقط لـ k، هي $k = i, i+1, \ldots, j-1$. ولأن وضع الأقواس الأمثل يجب أن يَستخدم قيمةً واحدة فقط من قيم k، فما علينا إلا أن ندقِّق فيها جميعها لإيجاد الأفضل. وبذلك يصبح تعريفنا العودي للتكلفة الصغرى لوضع أقواس الجداء $A_iA_{i+1}\cdots A_j$ كما يلي:

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \ , \\ \min\limits_{i \le k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & \text{if } i < j \ . \end{cases} \qquad (7.15)$$

تعطي القيم $m[i,j]$ تكاليفَ الحلول المثلى للمسائل الجزئية، ولكنها لا تعطي كل المعلومات التي نحتاج إليها لبناء حل أمثل. ونستعين على ذلك بأن نعرِّف $s[i,j]$ على أنها قيمة k التي نفرق عندها الجداء $A_iA_{i+1}\cdots A_j$ للحصول على وضع الأقواس الأمثل. أي إن $s[i,j]$ تساوي قيمة k بحيث $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$.

الخطوة 3: حساب التكاليف المثلى

عند هذه النقطة، من السهل كتابة خوارزمية عودية تعتمد على العلاقة العودية (7.15) لحساب التكلفة الصغرى $m[1,n]$ للجداء $A_1A_2\cdots A_n$. وحسبما رأينا في مسألة تقطيع القضبان، وكما سنرى في المقطع 3.15، فإن هذه الخوارزمية تستغرق زمنًا أسيًّا، وهو ليس أفضل من طريقة البحث الشامل لفحص كل طرق وضع الأقواس لحساب الجداء.

لاحظ أن لدينا عددًا قليلاً نسبيًّا من مسائل جزئية متمايزة: مسألة واحدة لكل خيار لـ i و j بحيث

يكون $1 \le i \le j \le n$، أو تعقيدًا كليًّا $\binom{n}{2} + n = \Theta(n^2)$. يمكن أن تصادف خوارزميةٌ عودية كلَّ مسألة جزئية عدةَ مرات في الفروع المختلفة لشجرتها العودية. إن خاصية تداخل المسائل الجزئية هذه هي سمة مميزة ثانية لقابلية تطبيق البرمجة الديناميكية (السمة المميزة الأولى هي البنية الجزئية المثلى).

عوضًا عن حساب الحل للعلاقة العودية (7.15) عوديًّا، فإننا نحسب التكلفة المثلى باستخدام نهج صعودي يعتمد جدولاً. (سنعرض النهج التُّزولي الموافق باستخدام الاستذكار في المقطع 3.15.)

وسننجز الطريقة الصعودية الجدولية في الإجراء MATRIX-CHAIN-ORDER الذي يظهر لاحقًا. يفترض هذا الإجراء أن بُعد المصفوفة A_i هو $p_{i-1} \times p_i$ حيث $i = 1, 2, \dots, n$. ودخل الإجراء هو المتتالية $p = \langle p_0, p_1, \dots, p_n \rangle$، حيث $p.length = n + 1$. ويستخدم الإجراء جدولاً مساعدًا $m[1..n, 1..n]$ لخزن التكاليف $m[i, j]$ ، وجدولاً مساعدًا آخر $s[1..n-1, 2..n]$ لتسجيل الدليل k الذي كان قد أنجز التكلفة المثلى في حساب $m[i, j]$. وسنستخدم الجدول s لبناء الحل الأمثل.

لتنجيز النهج الصعودي، علينا أن نحدد عناصر الجدول المُستَخدَمة في حساب $m[i, j]$. تبيّن المعادلة (7.15) أن التكلفة $m[i, j]$ لحساب جداء سلسلة المصفوفات المكون من $j - i + 1$ مصفوفة يعتمد فقط على تكاليف حساب جداء سلسلة مصفوفات عددها أقل من $j - i + 1$. أي لكل $k = i, i + 1, \dots, j - 1$، فإن المصفوفة $A_{i..k}$ هي جداء $k - i + 1 < j - i + 1$ مصفوفة، والمصفوفة $A_{k+1..j}$ هي جداء $j - k < j - i + 1$ مصفوفة. وبذلك، يجب أن تملأ الخوارزميةُ الجدولَ بقيم m على نحو يتوافق مع حل مسألة وضع الأقواس على سلسلة مصفوفات ذات طول متزايد. وفيما يتعلق بالمسألة الجزئية لوضع الأقواس على نحو أمثل لسلسلة المصفوفات $A_i A_{i+1} \cdots A_j$، نفترض أن حجم المسألة الجزئية هو طول السلسلة $j - i + 1$.

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length - 1
 2  let m[1..n, 1..n] and s[1..n - 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length.
 6      for i = 1 to n - l + 1
 7          j = i + l - 1
 8          m[i, j] = ∞
 9          for k = i to j - 1
10              q = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

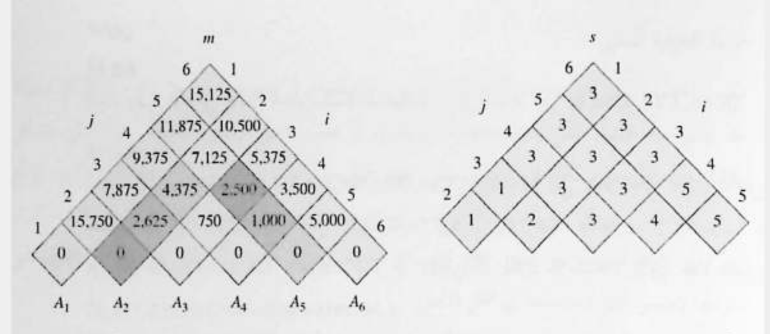
تَحسب الخوارزمية أولاً $m[i,i] = 0$ للقيم $i = 1,2,\dots,n$ (التكاليف الصغرى لسلاسل بطول 1) في السطرين 3–4. بعد ذلك تستخدم العلاقة التكرارية (7.15) لحساب $m[i, i+1]$ لقيم $i = 1, 2, \dots, n-1$ (التكاليف الصغرى لسلاسل بطول 2 $l = 2$) خلال التنفيذ الأول لحلقة **for** في السطور 5–13. وخلال التنفيذ الثاني للحلقة تحسب $m[i, i+2]$ لقيم $i = 1,2,\dots,n-2$ (التكاليف الصغرى لسلاسل بطول 3 $l = 3$)، وهكذا. في كل خطوة، تعتمد التكاليف $m[i, j]$ المحسوبة في السطور 10–13 فقط على عناصر الجدول $m[i, k]$ و $m[k+1, j]$ المحسوبة سلفًا.

يبيّن الشكل 5.15 هذه الإجرائية على سلسلة من $n = 6$ مصفوفات. ولأننا عرّفنا $m[i, j]$ فقط في حالة $i \le j$، فإننا نستخدم فقط جزء الجدول m الذي يقع فوق القطر تمامًا. يبيّن الشكل الجدول مدوّرًا لجعل القطر الرئيسي يبدو أفقيًا. جرى سرد سلسلة المصفوفات في الأسفل. وباستخدام هذا المخطط، يمكن إيجاد التكلفة الصغرى $m[i, j]$ لجداء السلسلة الجزئية من المصفوفات $A_iA_{i+1}\cdots A_j$ عند تقاطع الخط الذي يبدأ من A_i باتجاه الشمال الشرقي مع الخط الذي يبدأ من A_j باتجاه الشمال الغربي. يتضمن كل سطر أفقي في الجدول القيم من أجل سلاسل المصفوفات ذات الطول نفسه. يحسب الإجراء MATRIX-CHAIN-ORDER السطور من الأسفل إلى الأعلى ومن اليسار إلى اليمين ضمن كل سطر. وهو يحسب كل العناصر $m[i, j]$ باستخدام الجداء $p_{i-1}p_kp_j$ لقيم $k = i, i+1, \dots, j-1$ ولكل العناصر الجنوبية الغربية والجنوبية الشرقية من $m[i, j]$.

تبيّن معاينةٌ بسيطة لبنية الحلقات المتداخلة في MATRIX-CHAIN-ORDER أن زمن تنفيذ الخوارزمية هو $O(n^3)$. إذ هناك ثلاث حلقات متداخلة، وكل دليل حلقة (l و i و k) يأخذ $n-1$ قيمة على الأكثر. يُطلب إليك في التمرين 15.2-5 أن تبين أن زمن تنفيذ هذه الخوارزمية هو في الحقيقة $\Omega(n^3)$ أيضًا. تتطلب الخوارزمية $\Theta(n^2)$ مكانًا لخزن الجدولين m و s. وبذلك، فإن الخوارزمية MATRIX-CHAIN-ORDER أكثر فعالية بكثير من الطريقة التي زمنها أسي، والتي تحصي كل أوضاع الأقواس الممكنة وتفحص كلاً منها.

الخطوة 4: بناءُ حلٍّ أمثلَ

مع أن الخوارزمية MATRIX-CHAIN-ORDER تحدد العدد الأمثل للجداءات السلمية اللازمة لحساب جداء سلسلة مصفوفات، فهي لا تبيّن مباشرة كيف تجري عملية ضرب المصفوفات. لكن الجدول $s[1..n-1, 2..n]$ يزودنا بالمعلومات اللازمة للقيام بذلك. يسجل كل عنصر $s[i, j]$ قيمة k بحيث يفرّق الوضعُ الأمثل لأقواس الجداء $A_iA_{i+1}\cdots A_j$ هذا الجداءَ بين A_k و A_{k+1}. وبذلك نعلم أن جداء مصفوفات النهائي في حساب $A_{1..n}$ أمثليًا هو $A_{1..s[1,n]}A_{s[1,n]+1..n}$. ويمكن حساب جداءات المصفوفات السابقة عوديًا، لأن $s[1, s[1, n]]$ تحدد آخر جداء مصفوفات عند حساب $A_{1..s[1,n]}$ وتحدد $s[s[1, n]+1, n]$ آخر جداء مصفوفات عند حساب $A_{s[1,n]+1..n}$. يطبع الإجراء العودي التالي وضعًا أمثلَ للأقواس



الشكل 5.15 الجداول m و s محسوبة بالخوارزمية MATRIX-CHAIN-ORDER في حالة $n = 6$ وأبعاد المصفوفات التالية:

المصفوفة	A_1	A_2	A_3	A_4	A_5	A_6
أبعادها	30×35	35×15	15×5	5×10	10×20	20×25

جرى تدوير الجداول بحيث يظهر القطر الرئيسي أفقيًا. لا نستخدم إلا القطر الرئيسي والمثلث الذي فوقه في الجدول m، والمثلث العلوي فقط في الجدول s. العدد الأصغر للجداءات السلمية لضرب ست مصفوفات هو $m[1,6] = 15,125$. ومن العناصر الغامقة، تؤخذ الأزواج التي لها التظليل نفسه معًا في السطر 10 عند حساب

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + (35)(15)(20) = 13000 \, , \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + (35)(5)(20) = 7125 \, , \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + (35)(10)(20) = 11375 \end{cases}$$
$$= 7125 \, .$$

في $\langle A_i, A_{i+1} \cdots, A_j \rangle$، في حال أُعطينا الجدول s المحسوب بواسطة MATRIX-CHAIN-ORDER والدليلَيْن i و j. ويطبع الاستدعاءُ الأولي للإجرائية PRINT-OPTIMAL-PARENS$(s, 1, n)$ وضعًا أمثل لأقواس الجداء $\langle A_1, A_2, \dots, A_n \rangle$.

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2        print "A"i
3  else print "("
4        PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5        PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6        print ")"
```

في المثال المبيّن بالشكل 5.15 يَطبع الاستدعاءُ PRINT-OPTIMAL-PARENS$(s, 1, 6)$ الأقواسَ التالية: $((A_1(A_2A_3))((A_4A_5)A_6))$.

تمارين

1-2.15

أوجد وضع الأقواس الأمثل لجداء سلسلة مصفوفات، تتالي أبعادها هو: $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$.

2-2.15

أعطِ الخوارزمية العودية MATRIX-CHAIN-MULTIPLY(A, s, i, j) التي تنجز فعليًا الجداءَ الأمثل لسلسلة مصفوفات، إذا كان لدينا تتالي المصفوفات $\langle A_1, A_2, \ldots, A_n \rangle$، والجدول s المحسوب بالخوارزمية MATRIX-CHAIN-ORDER والدليلان i و j. (يمكن أن يكون الاستدعاء الأول لهذه الخوارزمية هو: MATRIX-CHAIN-MULTIPLY$(A, s, 1, n)$.)

3-2.15

استخدم طريقة التعويض لتبيّن أن حل العلاقة التكرارية (6.15) هو $\Omega(2^n)$.

4-2.15

صِف بيان المسألة الجزئية لضرب سلسلة المصفوفات، بسلسلة دخل طولها n. كم عدد العقد فيه؟ وكم عدد الوصلات فيه؟ وما هي هذه الوصلات؟

5-2.15

ليكن $R(i, j)$ عدد المرات التي نعود فيها إلى عنصر الجدول $m[i, j]$ عند حساب عناصر الجدول الأخرى في الاستدعاء MATRIX-CHAIN-ORDER. بيّن أن العدد الكلي للمرات التي نعود فيها إلى كامل الجدول هو:

$$\sum_{i=1}^{n} \sum_{j=i}^{n} R(i, j) = \frac{n^3 - n}{3} .$$

(*تلميح*: يمكن الاستفادة من المساواة (أ.3).)

6-2.15

بيّن أن وضع كامل الأقواس لتعبير من n عنصرًا يتطلب تمامًا $n - 1$ زوجًا من الأقواس.

3.15 عناصر البرمجة الديناميكية

مع أننا عملنا حتى الآن على مثالين على طريقة البرمجة الديناميكية، فربما ما زلتَ تتساءل أين يمكن تطبيق هذه الطريقة. فمن وجهة نظر هندسية نتساءل: متى يتعيَّن علينا البحث عن حلّ مسألة بالبرمجة الديناميكية؟ في هذا المقطع، سندرس المكوِّنَيْن الرئيسيين اللذين يجب أن يتوفرا في مسألة الأمثَلَة لكي يكون بالإمكان تطبيق البرمجة الديناميكية: بنية جزئية مُثلى، ومسائل جزئية متراكبة. سنعود أيضًا ونناقش بتفصيلٍ أكبر كيف يمكن أن يساعدنا الاستذكار memoization للإفادة من خاصية تراكب المسائل الجزئية في نهجٍ عودي نزولي.

البنية الجزئية المثلى

تكمن الخطوة الأولى -في حل مسألة أمثَلَة بالبرمجة الديناميكية- في توصيف بنية حلٍّ أمثل. تذكَّرْ أن مسألة ما، تُظهر ***بنيةً جزئيةً مثلى optimal substructure*** إذا تضمن الحل الأمثل للمسألة حلولاً مثلى لمسائل جزئية. فكلما أبدت مسألةٌ ما بنيةً جزئيةً مثلى، فهذا دليل جيد على إمكان تطبيق البرمجة الديناميكية. (ومع ذلك، فإن هذا يمكن أن يعني أيضًا إمكان تطبيق استراتيجية شرهة. انظر الفصل 16.) في البرمجة الديناميكية، نبني الحل الأمثل للمسألة من الحلول المثلى للمسائل الجزئية. ومن ثَمّ فعلينا التأكد أن مجال المسائل الجزئية التي ندرسها يتضمن تلك المستَخدَمة في الحل الأمثل.

اكتشفنا حتى الآن البنية الجزئية المثلى في كلتا المسألتين اللتين درسناهما في هذا الفصل. ففي المقطع 1.15، لاحظنا أن الطريقة المثلى لقطع قضيب طوله n (إذا كان هناك قطع أصلاً) تتطلب تقطيعًا أمثل لقطعتين نتجتا عن أول قطع. ولاحظنا في المقطع 2.15 أن وضع الأقواس الأمثل لـ $A_iA_{i+1}\cdots A_j$ الذي يفرق الجداء بين A_k و A_{k+1} يتضمن حلولاً مثلى لمسائل وضع الأقواس لكل من $A_iA_{i+1}\cdots A_k$ و $A_{k+1}A_{k+2}\cdots A_j$.

وستجد نفسك تتعقب نموذجًا مشتركًا في اكتشاف بنيةٍ جزئية مثلى:

1. سيتبيّن لك أن حل المسألة يتضمن إجراء عملية اختيار، مثل اختيار القَطْع الأولي في قضيب أو اختيار الدليل الذي نفرق عنده سلسلة المصفوفات. إن إجراء هذا الاختيار يبقي مسألةً جزئية أو أكثر يجب حلها.
2. ستفترض أنه في مسألة معينة، سيكون لديك الخيار الذي يقود إلى حلٍّ أمثل. لن يكون عليك أن تهتم بعدُ بكيفية تحديد هذا الخيار،إنك تفترض أنه قد أُعطي إليك.
3. إذا أُعطيت هذا الخيار، عليك أن تحدد أي المسائل الجزئية ستنشأ، وكيف ستوصّف أمثليًّا فضاء المسائل الجزئية الناتجة.
4. سيتبيّن لك أن حلول المسائل الجزئية المُستخدمة ضمن الحل الأمثل للمسألة يجب أن تكون هي نفسها مثلى، باستخدام تقنية "قُصَّ والصُق". وبإمكانك إجراء ذلك بافتراض أن كلاًّ من حلول المسائل الجزئية غير أمثلي، ثم الوصول إلى تناقض. وبوجهٍ خاص، سترى أنك "بقصّ" حلول المسائل الجزئية غير المثلى، و"لصق" الحل الأمثل، ستصل إلى حل أفضل للمسألة الأصلية، وبذلك تُناقض افتراضك بأنه كان لديك سابقًا حل أمثل. وإذا أعطى حلٌّ أمثلُ أكثرَ من مسألةٍ جزئية واحدة، كانت هذه المسائل الجزئية متشابهة جدًّا إلى درجة أنه يمكن، بقليل من الجهد، تعديل عملية القص واللصق لإحداها وتطبيقها على المسائل الأخرى.

ولتوصيف فضاء المسائل الجزئية، فإن القاعدة الذهبية تقول بأن نحاول جعل هذا الفضاء سهلاً قدر الإمكان، ثم توسيعه بقدر الحاجة إلى ذلك. على سبيل المثال، كان فضاء المسائل الجزئية الذي أخذناه بالحسبان في مسألة تقطيع القضبان يتضمن مسائلَ تقطيعِ أمثلَ لقضيبٍ طولُهُ i، لكل قيم الطول i. وكان هذا الفضاء جيد الأداء، ولم يكن هناك داعٍ لتجريب فضاء مسائل جزئية أكثر عمومية.

وبالعكس، افترض أننا حاولنا أن نَقْصُرَ فضاءَ المسائل الجزئية في مسألة جداء سلسلة مصفوفات على جداء مصفوفات من الشكل $A_1A_2 \cdots A_j$. كما في السابق، يجب أن يفرق وضع الأقواس الأمثل هذا الجداء بين المصفوفتين A_k و A_{k+1}، لقيمة ما k بحيث $1 \leq k < j$. وما لم نضمن أن k تساوي دائمًا $j-1$، سنجد أن لدينا مسائل جزئية من الشكل $A_1A_2 \cdots A_k$ و $A_{k+1}A_{k+2} \cdots A_j$، وأن المسألة الجزئية الأخيرة ليست من الشكل $A_1A_2 \cdots A_j$. وفي هذه المسألة، كان يلزمنا أن نسمح لمسائلنا الجزئية أن تتغير "من الطرفين"، أي أن نسمح لـ i و j بالتغيُّر في المسألة الجزئية $A_iA_{i+1} \cdots A_j$.

يمكن أن تتغير البنية الجزئية المثلى عبر مجالات المسألة بطريقتين:

1. ما هو عدد المسائل الجزئية التي يستخدمها الحل الأمثل للمسألة الأصلية، و
2. ما هو عدد الخيارات لدينا لتحديد المسألة أو المسائل الجزئية التي علينا استخدامها في الحل الأمثل.

في مسألة تقطيع القضبان، يَستخدم الحلُّ الأمثل لتقطيع قضيب طوله n مسألةً جزئية وحيدة (حجمها $n-i$)، ولكن علينا أن نأخذ بالحسبان n خيارًا لـ i لتحديد أيها يعطي حلاً أمثلَ. تمثّل مسألةُ جداء سلسلة المصفوفات للسلسلة الجزئية $A_iA_{i+1} \cdots A_j$ مثالاً له مسألتان جزئيتان و $j-i$ خيارًا. في حالة مصفوفة معلومةٍ A_k التي نفرق عندها الجداء، سيكون لدينا مسألتان جزئيتان: وضع الأقواس للجداء $A_iA_{i+1} \cdots A_k$ ووضعها للجداء $A_{k+1}A_{k+2} \cdots A_j$، وعلينا أن نحلّ كل منهما حلاً أمثلَ. وحالما نجد الحلول المثلى للمسائل الجزئية، نختار الدليل k من بين الـ $j-i$ دليلاً مُرشحًا.

وعلى نحو غير رسمي، يعتمد زمن تنفيذ خوارزمية البرمجة الديناميكية على جداء عاملَيْن: عدد المسائل الجزئية الكلية، وعدد الخيارات التي ننظر فيها لكل مسألة جزئية. ففي مسألة تقطيع القضبان، كان عدد المسائل الجزئية الكلي $\Theta(n)$ وعلينا أن نفحص n خيارًا على الأكثر لكلٍّ منها، وهذا يعطي زمنَ تنفيذ $O(n^2)$. وفي حالة جداء سلسلة المصفوفات، كان عدد المسائل الجزئية الكلية $\Theta(n^2)$ ولكل منها $n-1$ خيارًا على الأكثر، وبذلك يكون زمن التنفيذ $O(n^3)$ (فعليًا زمن التنفيذ $\Theta(n^3)$، وفقًا للتمرين 15.2-5).

يُعطي بيانُ المسائل الجزئية، عادة، طريقةً بديلةً لإنجاز التحليل نفسه؛ إذ توافق كلُّ عقدةٍ مسألةً جزئية، وخياراتُ أي مسألةٍ جزئية هي الوصلات المرتبطة بتلك المسألة الجزئية. تذكّر أنه في مسألة تقطيع القضبان، يتضمن بيان المسألة الجزئية n عقدة، و n وصلة على الأكثر لكل عقدة، وهذا يعطي زمنَ تنفيذ $O(n^2)$. وفي حالة مسألة جداء المصفوفات، لو كان علينا رسم بيان المسائل الجزئية، لتضمّن $\Theta(n^2)$ عقدة، ولكان

لكل عقدة درجة تساوي على الأكثر $n - 1$، وهذا يعطي ما مجموعه $O(n^3)$ عقدة ووصلة.

غالبًا ما تَستخدم البرمجةُ الديناميكيةُ البنيةَ الجزئية المثلى بطريقة صعودية. أي إننا نجد أولاً حلولاً مثلى لمسائل جزئية، وبعد حل المسائل الجزئية نجد حلاً أمثليًا للمسألة. ويستلزم إيجاد حل أمثل للمسألة، اختيار المسألة الجزئية التي سنستخدمها في حل المسألة من بين المسائل الجزئية. إن تكلفة حل المسألة تساوي عادة تكاليف المسائل الجزئية إضافةً إلى تكلفةٍ تُعزى مباشرة إلى الاختيار نفسه. على سبيل المثال، في مسألة تقطيع القضبان، قمنا أولاً بحل المسائل الجزئية لإيجاد الطرق المثلى لتقطيع القضبان ذات الطول i لقيم $i = 0, 1, ..., n - 1$ ثم حددنا أي المسائل الجزئية أعطت حلاً أمثل لقضيب طوله n باستخدام المعادلة (2.15). إن تكلفة الخيار نفسه هو الحد p_i في المعادلة (2.15). وفي مسألة جداء سلسلة المصفوفات، حددنا وضع الأقواس الأمثل للسلسلة الجزئية $A_i A_{i+1} \cdots A_j$، ثم اخترنا المصفوفة A_k التي نفرق الجداء عندها. أما التكلفة التي نعزوها إلى الاختيار نفسه فهي الحد $p_{i-1}p_k p_j$.

سندرس في الفصل 16 "الخوارزميات الشرهة"، التي تشابه كثيرًا البرمجةَ الديناميكية. وعلى وجه الخصوص، فإن للمسائل التي تنطبق عليها الخوارزمياتُ الشرهة بنيةً جزئية مثلى. ومن الفروق الرئيسية بين الخوارزميات الشرهة والبرمجة الديناميكية أنه عوضًا عن إيجاد الحلول المثلى للمسائل الجزئية أولاً، ثم الاختيار عن علمٍ، تقوم الخوارزمياتُ الشرهة أولاً باختيار "شره"– الخيار الذي يبدو أنه الأفضل في ذلك الوقت – ثم تَحلُّ المسألةَ الجزئيةَ الناتجة، دون الالتفات إلى حلّ كل المسائل الجزئية الأصغر المحتملة ذات الصلة. ومما يثير الدهشة فعلاً أن هذه الاستراتيجية تعمل بنجاح في بعض الأحيان!

نقاط دقيقة

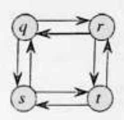
ينبغي ألاّ نفترض أن البنى الجزئية المثلى محقَّقة، في حين لا يكون الأمر كذلك. لندرس المسألتين التاليتين، اللتين لدينا فيهما بيان موجَّه $G = (V, E)$ والعقد $u, v \in V$.

أقصر مسار غير مثقَّل[3]: أوجد مسارًا من u إلى v مؤلَّفًا من أقل عددٍ من الوصلات. يجب أن يكون هذا المسار بسيطًا، لأن حذف حلقة cycle من المسار يولِّد طريقًا بوصلات أقل.

أطول مسار بسيط غير مثقَّل: أوجد مسارًا بسيطًا من u إلى v يتكون من معظم الوصلات. نحتاج إلى إدخال مطلب البساطة لأننا بغير ذلك يمكن أن نجتاز حلقةً بقدر ما نريد من المرات لننشئ مساراتٍ بعدد وصلات لا على التعيين.

---

[3] نستخدم التعبير "غير مثقَّل unweighted" لتمييز هذه المسألة عن تلك التي توجد أقصر مسار مع وصلات مثقَّلة weighted، التي سنتاولها في الفصلين 24 و 25. يمكننا توظيف تقنية البحث عرضًا أولاً breadth-first search technique المستخدمة في الفصل 22 لحل المسألة غير المثقلة.



الشكل 6.15 بيان موجّه يبين أنه لا توجد بنية جزئية مثلى لمسألة إيجاد أطول مسار بسيط في بيانٍ موجَّهٍ غير مثقَّل. المسار $q \to r \to t$ هو أطول مسار بسيط من q إلى t، إلا أن المسار الجزئي $q \to r$ ليس أطول مسار بسيط من q إلى r، كما أن المسار الجزئي $r \to t$ ليس أطول مسار بسيط من r إلى t.

تُبدي مسألة أقصر طريق غير مثقَّل بنيةً مثلى كما يلي: افترض أن $u \neq v$، بحيث لاتكون المسألة تافهة. بعدها، يجب أن يتضمن كل مسار p من u إلى v عقدة وسيطة ولتكن w (لاحظ أن w يمكن أن تكون u أو v). وبذلك يمكننا تقسيم المسار $u \overset{p}{\to} v$ إلى المسارات الجزئية $u \overset{p_1}{\to} w \overset{p_2}{\to} v$. ومن الواضح أن عدد الوصلات p يساوي مجموع عدد الوصلات في p_1 و p_2. ندّعي أنه إذا كانت p مسارًا أمثل (أقصر مسار) من u إلى v فإن p_1 يجب أن تكون أقصر مسار من u إلى w. لماذا؟ نستخدم طريقة المحاكمة "قصّ والصق": فلو كان هناك مسار آخر، وليكن p'_1، من u إلى w بوصلات أقل من p_1، لأمكننا إذًا قص p_1 ولصق p'_1 لتوليد مسار $u \overset{p'_1}{\to} w \overset{p_2}{\to} v$ بوصلات أقل من p، وهذا يناقض كون p مثاليًا. وبالمثل، يجب أن تكون p_2 أقصر مسار من w إلى v. وبذلك، يمكن إيجاد أقصر مسار من u إلى v بأخذ كل العقد الوسيطة w بالاعتبار، وإيجاد أقصر مسار من u إلى w وأقصر مسار من w إلى v، واختيار عقدة وسيطة w تعطي أقصرَ مسار كلي. نستخدم في المقطع 2.25 شكلاً مختلفًا لهذه الملاحظة المتعلقة بالبنية الجزئية المثلى لإيجاد أقصر مسارٍ بين كل زوجين من العُقد على بيانٍ موجَّه مثقَّل.

قد يكون من المغري افتراض أن مسألة إيجاد أطول مسار بسيط غير مثقَّل تبدي بنية جزئية مثلى أيضًا. وبعد هذا، إذا حلَّلنا أطول مسار بسيط $u \overset{p}{\to} v$ إلى مسارَيْن جزئيين $u \overset{p_1}{\to} w \overset{p_2}{\to} v$، عندها أليس من الواجب أن تكون p_1 أطول مسار بسيط بين u و w، وأن تكون p_2 أطول *مسار* بسيط بين w و v؟ الجواب لا! يبيّن الشكل 6.15 مثالاً على ذلك. لنأخذ المسار $q \to r \to t$ وهو أطول مسار بسيط من q إلى t. هل $q \to r$ هي أطول مسار بسيط من q إلى r؟ الجواب لا، لأن المسار $q \to s \to t \to r$ مسار بسيط أطول منه. هل $r \to t$ هي أطول مسار بسيط من r إلى t؟ الجواب لا أيضًا، لأن المسار $r \to q \to s \to t$ مسار بسيط أطول منه.

يبيّن هذا المثال أنه في حالة أطول مسارات بسيطة، لا تفتقر المسألة إلى بنية جزئية مثلى فقط؛ وإنما لا يمكننا بالضرورة تجميع حل "شرعي" للمسألة من حلول لمسائل جزئية. إذا ضَمَمْنا المسارَيْن البسيطين الطويلين: $q \to s \to t \to r$ و $r \to q \to s \to t$ حصلنا على $q \to s \to t \to r \to q \to s \to t$

وهو مسار ليس بسيطًا. في الواقع، لا يبدو أن مسألة إيجاد أطول مسار بسيط غير مثقَّل تمتلك أي نوع من البنى الجزئية المثلى. ولم يُعْثَر قطُّ على أي خوارزمية تعتمد على برمجة ديناميكية فعالة لحل هذه المسألة. والحقيقة أن هذه المسألة تامة غير حدودية NP-complete، وهذا يعني – كما سنرى في الفصل 34 – أنه ليس من المحتمل أن نجد طريقًا لحلها بزمن كثير حدودي.

لماذا كانت البنية الجزئية لأطول مسار بسيط مختلفة جدًّا عن البنية الجزئية لأقصر مسار؟ ومع أن حلَّ مسألةٍ لأطول المسارات وأقصرها يتطلب استخدام مسألتَيْن جزئيتين، فإن المسائل الجزئية المستَخدَمة في إيجاد أطول مسار بسيط ليست مسائل ***مستقلة independent*** بعضها عن بعض، في حين أنها مستقلة في حالة أقصر المسارات. ماذا نقصد بكون المسائل الجزئية مستقلة؟ نقصد بذلك أن حل إحدى المسائل الجزئية لا يؤثر في حل مسألة جزئية أخرى للمسألة الأصلية نفسها. على سبيل المثال، في الشكل 6.15 لدينا مسألة إيجاد أطول مسار بسيط من q إلى t ولها مسألتان جزئيتان: إيجاد أطول مسار بسيط من q إلى r ومن r إلى t. نختار للمسألة الجزئية الأولى المسار: $q \to s \to t \to r$، وبذلك نكون قد استعملنا أيضًا العقدتين s و t. ولن يكون بإمكاننا استعمالهما بعد الآن في حل المسألة الجزئية الثانية، لأن ضم الحلين للمسألتين الجزئيتين سيعطي مسارًا غير بسيط. فإذا تعذَّر علينا استعمال العقدة t في المسألة الثانية، عندها لن يكون بإمكاننا حل المسألة على الإطلاق، إذ يتطلَّب الأمرُ أن تكون t على المسار الذي نوجده، وهي ليست العقدة التي نربط بها حَلَّي المسألتين الجزئيتين (لأن عقدة الربط هي r). ولأننا استخدمنا العقدتين s و t في حل إحدى المسألتين الجزئيتين، فإننا لا نستطيع استخدامهما في حل المسألة الجزئية الأخرى. ولكن علينا استخدام إحدى العقدتين على الأقل لحل المسألة الجزئية الأخرى، وعلينا استخدامهما معًا لحل المسألة أمثليًّا. وهكذا نقول إن هاتين المسألتين الجزئيتين ليستا مستقلتين. وبالنظر إلى ذلك بطريقة أخرى، فإن استخدام الموارد في حل إحدى المسائل الجزئية (هذه الموارد هي العقد) يجعلها غير متاحة للمسألة الجزئية الأخرى.

لماذا إذن تكون المسائل الجزئية مستقلة بعضها عن بعض عند إيجاد أقصر مسار؟ الجواب هو أن هذه المسائل الجزئية، بطبيعتها، لا تتشارك في الموارد. إننا ندّعي أنه لو وجِدَت عقدة w على أقصر مسار p من u إلى v، نستطيع عندها أن نربط بين *أي* أقصر مسار $u \overset{p_1}{\to} w$ وأقصر مسار $w \overset{p_2}{\to} v$ لتوليد أقصر مسار من u إلى v. إننا متأكدون من أنه لن تظهر عقدة أخرى غير w في كلا المسارين p_1 و p_2. لماذا؟ لنفترض أن عقدة ما $x \neq w$ تظهر في كلا المسارين p_1 و p_2 بحيث يمكننا تحليل p_1 إلى $u \overset{p_{ux}}{\to} x \to w$ و p_2 إلى $w \to x \overset{p_{xv}}{\to} v$. في البنية الجزئية المثلى لهذه المسألة، يكون للمسار p وصلات بعدد وصلات p_1 و p_2 معًا؛ وليكن عدد وصلات p مساويًا e. لننشئ الآن المسار $p' = u \overset{p_{ux}}{\to} x \overset{p_{xv}}{\to} v$ من u إلى v. ولأننا أزلنا المسارين من x إلى w، ومن w إلى x، ولكل منهما وصلة على الأقل، فإن المسار p' يتضمن $e - 2$ وصلة على الأكثر، وهذا يناقض الفرض بأن p هو أقصر مسار. وبذلك، نحن متأكدون أن المسائل الجزئية لمسألة أقصر مسار هي مسائل مستقلة.

إن المسألتين اللتين درسناهما في المقطعين 1.15 و 2.15 لهما مسائل جزئية مستقلة. وفي حالة جداء سلسلة مصفوفات، تتمثل المسائل الجزئية في ضرب سلاسل جزئية $A_i A_{i+1} \cdots A_k$ و $A_{k+1} A_{k+2} \cdots A_j$. هذه السلاسل الجزئية منفصلة، بحيث لا يمكن لأي مصفوفة أن تُتضمَّن في الجداءين معًا. وفي مسألة تقطيع القضبان، لتحديد أفضل طريق لتقطيع قضيب طوله n، ننظر في أفضل طرق تقطيع قضبان أطوالها i للقيم $i = 0, 1, \ldots, n-1$. ولما كان الحل الأمثل للمسألة ذات الطول n يتضمن واحدًا فقط من حلول هذه المسائل الجزئية (بعد أن نكون قد قطعنا أول قطعة)، فلن يكون استقلالُ الحلول الجزئية نقطةَ خلاف.

مسائل جزئية متراكبة

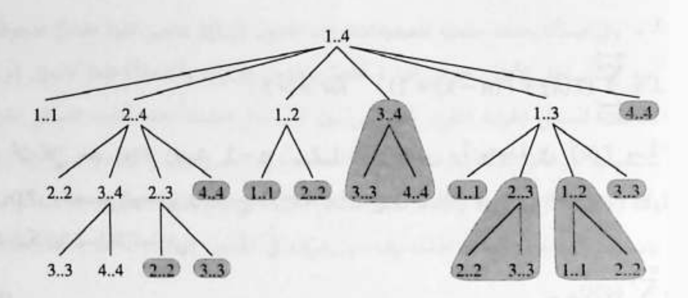
إن المكوّن الثاني الذي يجب أن تتضمنه مسألة الأمثَلَة لتكون البرمجة الديناميكية قابلة للتطبيق في حلها هو أن فضاء المسائل الجزئية يجب أن يكون "صغيرًا"، بمعنى أن الخوارزمية العودية التي تحل المسألة تحل المسائل الجزئية نفسها عدة مرات عوضًا عن توليد مسائل جزئية جديدة باستمرار. إن العدد الكلي للمسائل الجزئية المتمايزة هو، نموذجيًّا، كثير حدود في حجم المداخل. وحين تعود خوارزمية عودية إلى المسألة نفسها مرات عديدة في الزمن نقول إن لمسألة الأمثلَة ***مسائل جزئية متراكبة***[4] ***overlapping subproblems***. وبالمقابل، فإن المسألة التي يناسبها نهج "فرِّق-تسُد" تولِّد عادةً مسائلَ جديدة عند كل خطوة من العودية. وتستفيد خوارزميات البرمجة الديناميكية عادةً من تراكب المسائل الجزئية بحلّ كل مسألة جزئية مرةً واحدة، ثم تخزين الحل في جدول يمكن البحث عنه عند الحاجة، باستخدام زمن ثابت للبحث في الجدول.

في المقطع 1.15، درسنا باختصار كيف يُحْدِث حلٌّ عَوْديٌّ لمسألة تقطيع القضبان استدعاءات كثيرةَ العدد أسيًّا لإيجاد حلولٍ لمسائل جزئية أصغر. يخفّض حلُّنا بالبرمجة الديناميكية الخوارزمية العودية بزمنٍ أسي إلى زمن تربيعي.

ولبيان خاصية المسائل الجزئية المتراكبة بتفصيل أكبر، ندرس ثانية مسألة جداء سلسلة المصفوفات. بالعودة ثانية إلى الشكل 5.15، لاحظ أن الإجرائية MATRIX-CHAIN-ORDER تبحث تكراريًّا عن حل المسائل الجزئية في السطور الدنيا عند البحث عن حل مسائل جزئية في السطور العليا؛ فهي على سبيل المثال تشير إلى العنصر $m[3,4]$ أربع مرات: أثناء حساب $m[2,4]$ و $m[1,4]$ و $m[3,5]$ و $m[3,6]$. ولو أعدنا حساب $m[3,4]$ في كل مرة، عوضًا عن البحث عنها، لازداد زمن التنفيذ زيادةً مثيرة. ولكي نرى كيف

---

[4] قد يبدو غريبًا أن تعتمد البرمجة الديناميكية على كون المسائل الجزئية مستقلة ومتراكبة. ومع أن هذه المتطلبات قد تبدو متناقضة، فهي توصِّف مصطلحَيْن مختلفين، لا نقطتين على المحور نفسه. نقول عن مسألتين جزئيتين للمسألة نفسها إنهما مستقلتان إذا لم تتشاركا بالموارد. ونقول إنهما متراكبتان إذا كانت هذه المسائل الجزئية هي فعلاً المسائل الجزئية ذاتها التي نصادفها كمسائل جزئية لمسائل مختلفة.



الشكل 7.15 شجرة العودية لحساب RECURSIVE-MATRIX-CHAIN$(p, 1, 4)$. تتضمن كل عقدة الموسطات i و j. نستعيض عن الحسابات التي تُنجَز في الشجرة الفرعية المظللة بالبحث مرة واحدة فقط في MEMOIZED-MATRIX-CHAIN.

يحدث ذلك، ننظر في الإجرائية العودية التالية (غير الفعالة) التي تحدد $m[i,j]$، وهو أدنى عدد لعمليات الجداء السلمي الضرورية لحساب جداء سلسلة المصفوفات $A_{i..j} = A_i A_{i+1} \cdots A_j$. وتعتمد الخوارزمية مباشرة على العلاقة التكرارية (7.15).

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
1  if i == j
2      return 0
3  m[i, j] = ∞
4  for k = i to j − 1
5      q = RECURSIVE-MATRIX-CHAIN(p, i, k)
           + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
           + p_{i−1} p_k p_j
6      if q < m[i, j]
7          m[i, j] = q
8  return m[i, j]
```

يبين الشكل 7.15 شجرة العودية التي ينتجها استدعاء الخوارزمية RECURSIVE-MATRIX-CHAIN$(p, 1, 4)$. جرى وضع لصيقة على كل عقدة بقيم الموسطات i و j. لاحظ أن بعض أزواج القيم تَرِد عدة مرات.

والحقيقة أن بإمكاننا إثبات أن الزمن اللازم لحساب $m[1,n]$ بواسطة هذه الإجرائية العودية هو على الأقل زمن أسي في n. لنُشِرْ بـ $T(n)$ إلى الزمن الذي تستغرقه الإجرائية RECURSIVE-MATRIX-CHAIN لتحديد وضع الأقواس الأمثل لسلسلة من n مصفوفة. ولما كان تنفيذ السطور 1–2 والسطور 6–7 يستغرق الواحد منها وحدة زمنية واحدة على الأقل، وكذلك يستغرق الضرب في السطر 5، فإن فحص الإجرائية يعطي التتالي:

$$T(1) \geq 1 ,$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \qquad \text{for } n > 1 .$$

لاحظ أن كلَّ حد $T(i)$ (حيث $i = 1, 2, \ldots, n-1$) يَظهر مرةً واحدة في $T(k)$ ومرةً واحدة في $T(n-k)$، وبتجميع الوحدان في الجمع ($n-1$ مرة) مع 1 في المقدمة خارج المجموع، يمكننا إعادة كتابة العلاقة التكرارية السابقة كما يلي:

$$T(n) \geq 2 \sum_{i=1}^{n-1} T(i) + n . \tag{8.15}$$

سنثبت أن $T(n) = \Omega(2^n)$ بطريقة التعويض. وسنبيّن، بوجه خاص، أن $T(n) \geq 2^{n-1}$ لكل قيم $n \geq 1$. والقاعدة سهلة، لأن $T(1) \geq 1 = 2^0$. وبالاستقراء عندما يكون $n \geq 2$ لدينا:

$$\begin{aligned} T(n) &\geq 2 \sum_{i=1}^{n-1} 2^{i-1} + n \\ &= 2 \sum_{i=0}^{n-2} 2^i + n \\ &= 2(2^{n-1} - 1) + n \qquad \text{((من المعادلة (أ.5))} \\ &= 2^n - 2 + n \\ &\geq 2^{n-1} \end{aligned}$$

وهذا يتمم البرهان. وبذلك، يكون مقدار العمل الكلي المنجَز بالاستدعاء RECURSIVE-MATRIX-CHAIN$(p, 1, n)$ أسيًّا في n على الأقل.

قارنْ هذه الخوارزميةَ العودية التُّزولية بخوارزمية البرمجة الديناميكية الصعودية تجدْ أن الأخيرة أكثر فعالية لأنها تستفيد من خاصية تراكب المسائل الجزئية، وأن لمسألة جداء المصفوفات $\Theta(n^2)$ مسألة جزئية متمايزة فقط، وتَحُلُّ خوارزميةُ البرمجة الديناميكية كلَّ مسألةٍ منها مرةً واحدة فقط. من جهة أخرى، فإن على الخوارزمية العودية أن تعيد حل كل مسألة جزئية في كل مرة تظهر فيها في شجرة العودية. وكلما تضمنت شجرة العودية، لحل مسألة عودية طبيعية، المسألةَ الجزئية نفسها تكراريًّا، وكان العدد الكلي للمسائل الجزئية المتمايزة صغيرًا، فإن البرمجة الديناميكية تستطيع تحسين الفعالية، ويكون هذا التحسين أحيانًا مثيرًا.

إعادة إنشاء حل أمثل

كثيرًا ما نخزّن خياراتنا لكل مسألة جزئية في جدول، بحيث لا يترتب علينا إعادة إنشاء هذه المعلومات من التكاليف التي خزناها.

وفي مسألة جداء سلسلة المصفوفات، فإن الجدول $s[i,j]$ يختصر علينا مقدارًا ملحوظًا من العمل حين نعيد إنشاء الحل الأمثل. افترض أننا لم نحتفظ بالجدول $s[i,j]$، وأننا ملأنا فقط الجدول $m[i,j]$ الذي يحتوي تكاليف المسائل الجزئية المثلى. نختار من بين الـ $j-i$ احتمالاً عند تحديد المسائل الجزئية التي علينا أن نستعملها في الحل الأمثل لوضع الأقواس للجداء $A_iA_{i+1}\cdots A_j$، و $j-i$ ليس ثابتًا. لذلك، سنستغرق وقتًا $\Theta(j-i)=\omega(1)$ لإعادة إنشاء المسائل الجزئية التي وقع عليها اختيارنا حلاً لمسألةٍ معطاة. وبخزن دليل المصفوفة التي نفرق عندها الجداء $A_iA_{i+1}\cdots A_j$، في الجدول $s[i,j]$، يمكننا إعادة إنشاء كل الخيارات في زمن $O(1)$.

الاستذكار

مثلما رأينا في مسألة تقطيع القضبان، ثمة نهج بديل عن البرمجة الديناميكية غالبًا ما يوفِّر فعالية النهج الصعودي للبرمجة الديناميكية مع الاحتفاظ باستراتيجية نزولية. تكمن الفكرة في ***استذكار memoize*** الخوارزمية العودية الطبيعية غير الفعالة. ومثلما هو الحال في النهج الصعودي، فإننا نحتفظ بجدول يتضمن حلول المسائل الجزئية، إلا أن بنية التحكم لملء الجدول هي أشبه بالخوارزمية العودية.

تحتفظ خوارزمية الاستذكار العودية بعنصر في جدولٍ لحلّ كلّ مسألةٍ جزئية. يتضمن كلُّ عنصر للجدول مبدئيًّا قيمةً خاصة تشير إلى أن علينا ملء هذا العنصر. وحين نصادف المسألة الجزئية أول مرة، أثناء نَشْر الخوارزمية العودية، فإننا نحسب حلَّها ثم نخزنه في الجدول. وفي كلِّ مرةٍ نتعرَّض فيها لاحقًا لتلك المسألة الجزئية، فإننا ببساطة نبحث عن قيمتها المخزنة في الجدول ونعيدها إلى الخوارزمية.[5]

وفيما يلي نسخة مستَذكر من RECURSIVE-MATRIX-CHAIN. لاحظ مواضع الشبه مع الطريقة المستَذكَرة النُّزولية لمسألة تقطيع القضبان.

```
MEMOIZED-MATRIX-CHAIN(p)
1  n = p.length - 1
2  let m[1..n, 1..n] be a new table
3  for i = 1 to n
4      for j = i to n
5          m[i,j] = ∞
6  return LOOKUP-CHAIN(m, p, 1, n)
```

[5] يفترض هذا النهج سلفًا أننا نعلم مجموعة كل موسطات المسائل الجزئية الممكنة، وأن العلاقة بين مواقع الجدول والمسائل الجزئية موطدة ومعروفة. وثمة نهج آخر، أكثر عمومية، وهو استذكار القيم باستخدام التهشير (دالة البصمة) مع موسطات المسائل الجزئية كمفاتيح.

```
LOOKUP-CHAIN(m, p, i, j)
1  if m[i, j] < ∞
2       return m[i, j]
3  if i == j
4       m[i, j] = 0
5  else for k = i to j − 1
6       q = LOOKUP-CHAIN(m, p, i, k)
             + LOOKUP-CHAIN(m, p, k + 1, j) + p_{i−1} p_k p_j
7       if q < m[i, j]
8            m[i, j] = q
9  return m[i, j]
```

يحتفظ الإجراء MEMOIZED-MATRIX-CHAIN شأنَ الإجراء MATRIX-CHAIN-ORDER، بجدول $m[1..n, 1..n]$ للقيم المحسوبة لـ $m[i, j]$، وهو أصغر عدد للجداءات السلمية اللازمة لحساب المصفوفة $A_{i..j}$. يتضمن كل عنصر للجدول مبدئيًا القيمة ∞ لتشير إلى أن علينا ملء العنصر. فإذا جرى استدعاء LOOKUP-CHAIN(m, p, i, j)، وكانت قيمة $m[i, j]$ أصغر من ∞ في السطر 1، يعيد الإجراءُ ببساطة التكلفة المحسوبة سابقًا $m[i, j]$ (في السطر 2)، وإلا، يجري حساب التكلفة من الإجراء RECURSIVE-MATRIX-CHAIN وتخزن في $m[i, j]$ وتُعاد هذه التكلفة. وهكذا يعيد الإجراءُ LOOKUP-CHAIN(m, p, i, j) دائمًا القيمةَ $m[i, j]$، ولكنه يحسبها فقط في المرة الأولى التي يُستدعى فيها LOOKUP-CHAIN مع هاتين القيمتين المعيَّنتين لـ i و j.

يبيّن الشكل 15-7 كيف توفر الخوارزمية MEMOIZED-MATRIX-CHAIN الزمنَ مقارنةً بالخوارزمية RECURSIVE-MATRIX-CHAIN. تمثّل الشجراتُ الجزئيةُ المظللةُ القيمَ التي نبحث عنها في الجدول بدلاً من حسابها.

وكما هو الحال في خوارزمية البرمجة الديناميكية MATRIX-CHAIN-ORDER الصعودية، فإن الإجراء MEMOIZED-MATRIX-CHAIN ينفذ في زمن $O(n^3)$. ويُنفَّذ السطر 5 في MEMOIZED-MATRIX-CHAIN بزمن $\Theta(n^2)$. يمكننا تصنيف استدعاءات LOOKUP-CHAIN في نوعين:

1. استدعاء يكون فيه $m[i, j] = \infty$، فتُنفَّذ السطور 3–9 و
2. استدعاء يكون فيه $m[i, j] < \infty$، بحيث تعيد الخوارزميةُ LOOKUP-CHAIN ببساطةٍ القيمةَ في السطر 2.

ثمة $\Theta(n^2)$ استدعاءً من النوع الأول؛ واحد لكل عنصر في الجدول. تجري جميع الاستدعاءات من النوع الثاني باعتبارها استدعاءات عودية لاستدعاءات من النوع الأول. وكلما أجرت الخوارزميةُ LOOKUP-CHAIN استدعاءاتٍ عوديةً، فهي تُجري $O(n)$ استدعاءً منها. لذلك ثمة ما مجموعُهُ $O(n^3)$ استدعاءً من النوع الثاني.

وكلُّ استدعاءٍ من النوع الثاني يستغرق $O(1)$ من الزمن، في حين يستغرق كلُّ استدعاءٍ من النوع الأول زمنًا $O(n)$ مضافًا إليه الزمن المستغرَق في استدعاءاته العودية. فيكون إجماليُّ الزمن إذن $O(n^3)$. وبذلك يحوِّل الاستذكارُ خوارزميةً زمنُها $\Omega(2^n)$ إلى خوارزميةٍ زمنُها $O(n^3)$.

وخلاصة القول، يمكن حل مسألة جداء سلسلة مصفوفات إما بخوارزمية استذكار نزولية وإما بخوارزمية برمجة ديناميكية صعودية بزمن $O(n^3)$. وتستفيد كلتا الطريقتين من خاصية تراكب المسائل الجزئية. يوجد بالمجموع $\Theta(n^2)$ مسألة جزئية متمايزة فقط. وتَحسب كلٌّ من الطريقتين الحلَّ لكل مسألةٍ جزئية مرةً واحدةً فقط. وبدون الاستذكار، تُنفَّذ الخوارزميةُ العودية الطبيعية بزمنٍ أُسِّي، لأن المسائل الجزئية يُعاد حلها مرة بعد مرة (تكراريًّا).

وبصفةٍ عامة، إذا كان علينا حل جميع المسائل الجزئية مرةً واحدةً على الأقل، فإن خوارزمية البرمجة الديناميكية الصعودية تتفوَّق عادةً على خوارزمية الاستذكار النُّزولية بعامل ثابت، إذ لا يوجد للخوارزمية الصعودية عبءٌ إضافي للعودية، وعبء الاحتفاظ بالجدول هنا أقل من حالة الاستذكار. زِدْ على ذلك أن ثمة مسائل يمكن معها الاستفادة من النموذج النظامي للنفاذ إلى الجدول في خوارزمية البرمجة الديناميكية لخفض متطلبات الزمن أو الفضاء (الذاكرة) أكثر فأكثر. وبدلاً من ذلك، إذا لم يكن علينا حل بعض المسائل الجزئية في فضاء المسائل الجزئية على الإطلاق، فإن طريقة الاستذكار أفضل لأنها تحل فقط المسائل الجزئية المطلوبة حتمًا.

تمارين

1-3.15

أي الطريقتين أكثر فعاليةً لتحديد العدد الأمثل للجداءات في مسألة جداء سلسلة مصفوفات: عَدُّ كل طرق وضع أقواس الجداء وحساب عدد الجداءات لكل منها، أم تنفيذ خوارزمية RECURSIVE-MATRIX-CHAIN؟ علِّل إجابتك.

2-3.15

ارسم شجرة العودية للإجرائية MERGE-SORT من المقطع 3.2-1 لصفيفة من 16 عنصرًا. بيّن لماذا لا يكون الاستذكار فعالاً في تسريع خوارزمية فرِّق-تسد جيدة مثل MERGE-SORT.

3-3.15

ادرس نموذجًا لمسألة جداء سلسلة مصفوفات، الغرض منها وضع الأقواس لمتتالية مصفوفات لجعل عدد الجداءات السلمية اللازمة أعظميًّا عوضًا لا أصغريًّا. هل تبدي هذه المسألة بنية جزئية مثلى؟

4-3.15

ذكرنا سابقًا أننا، في البرمجة الديناميكية، نَحلُّ أولاً المسائلَ الجزئية، ثم نختار منها المسائل التي يجب استخدامها في الحل الأمثل للمسألة. ترى الأستاذة Capulet أنه ليس من الضروري دائمًا حل جميع المسائل الجزئية لإيجاد

الحل الأمثل. وتقترح أنه يمكن إيجاد الحل الأمثل لمسألة جداء سلسلة مصفوفات دائمًا باختيار المصفوفة A_k التي يجب عندها تفريق الجداء الجزئي $A_iA_{i+1}\cdots A_j$ (باختيار k التي تجعل المقدار $p_{i-1}p_kp_j$ أصغريًّا) *قبل* حل المسائل الجزئية. أوجد مثالاً على مسألة جداء سلسلة مصفوفات بحيث يكون لهذا النهج الشره حلاً أمثلَ جزئيًّا.

5-3.15

افترض أننا في مسألة تقطيع القضبان في المقطع 1.15، لدينا أيضًا القيد l_i على عدد القطع التي طولها i التي يُسمَح لنا بإنتاجها، حيث $i = 1, 2, \ldots, n$. بيّن أن خاصية البنية الجزئية المثلى الموصّفة في المقطع 1.15 لم تعد محققة.

6-3.15

تخيل أنك تريد تبديل عملة نقدية. إنك تدرك أنه عوضًا عن تبديل عملة مباشرة بأخرى قد يكون من الأفضل أن تُجري سلسلةً من التبادلات التجارية باستخدام عملات أخرى، بحيث تنتهي بالعملة التي تريد. افترض أنه يمكنك المتاجرة بـ n عملة مختلفة مرقمة $1, 2, \ldots, n$، حيث تبدأ بالعملة 1 وتنتهي بالعملة n. يُعطى سعرُ الصرف r_{ij} لكل زوج i و j من العملات، وهذا يعني أنك إذا بدأت بـ d وحدة من العملة i، فإنك ستبادلها بـ dr_{ij} وحدة من العملة j. قد تستلزم عمليةُ الصرف عمولةً تعتمد على عدد مرات الصرف التي تجريها. ليكن c_k مقدار العمولة التي تتحمَّلها حين تجري k عملية صرف. بيّن أنه إذا كان $c_k = 0$ للقيم $k = 1, 2, \ldots, n$، فإن مسألة إيجاد أفضل متتالية للتبادلات من العملة 1 إلى العملة n تُظهر بنيةً جزئية مثلى. ثم بيّن أنه إذا كانت العمولات c_k قيمًا اعتباطية، فإن مسألة إيجاد أفضل متتالية للتبادلات من العملة 1 إلى العملة n لا تُظهر بالضرورة بنيةً جزئية مثلى.

4.15 أطول متتالية جزئية مشتركة

كثيرًا ما تحتاج التطبيقاتُ البيولوجية مقارنةَ سلسلتَي DNA لكائنَيْن مختلفين (أو أكثر). تتكون جديلة DNA من متتالية جزئيات تسمى ***الأسس bases***، حيث الأسس الممكنة هي: الأدينين والغوانين والسيتوزين والتايمين. فإذا مثلنا كلاً من هذه الأسس بالحرف الأول من اسمها اللاتيني، أمكننا التعبير عن جديلة DNA بمتتالية محارف من المجموعة المنتهية $\{A, C, G, T\}$ (انظر الملحق-ت لتعريف متتالية محارف). على سبيل المثال، يمكن أن تكون جديلة الـ DNA لأحد الكائنات الحية $S_1 =$ ACCGGTCGAGTGCGCGGAAGCCGGCCGAA، في حين يمكن أن تكون لكائن حي آخر $S_2 =$ GTCGTTCGGAATGCCGTTGCTCTGTAAA. ومن دواعي مقارنة جديلتَي DNA معرفة مدى "التشابه" بينهما، باعتباره قياسًا لمدى قرابة هذين الكائنين. يمكننا تعريف التشابه، ونعرّفه فعلاً، بعدة طرق

مختلفة. على سبيل المثال، يمكن أن نقول عن جديلتَي DNA إنهما متشابهتان إذا كانت إحداهما متتالية محارف جزئية من الأخرى. (يناقش الفصل 32 خوارزميات لحل هذه المسألة.) في مثالنا ليست S_1 ولا S_2 متتالية محارف جزئية من الأخرى. بل يمكننا القول عن جديلتين إنهما متشابهتان إذا كان عدد التغييرات اللازمة لتحويل إحدى الجديلتين إلى الأخرى صغيرًا (تبحث المسألة 15-5 في هذا المفهوم). وثمة طريقة أخرى لقياس التشابه بين جديلتَين S_1 و S_2 تتمثل بإيجاد جديلة ثالثة S_3 تظهر أسسها في كل من S_1 و S_2؛ ويجب أن تظهر هذه الأسس بالترتيب نفسه، ولكن ليس بالضرورة على التتالي. وكلما كانت الجديلة S_3 أطول كان التشابه بين S_1 و S_2 أكبر. وفي مثالنا، أطول متتالية S_3 هي GTCGTCGGAAGCCGGCCGAA.

يمكن صوغ هذا المفهوم الأخير للتشابه اصطلاحًا على أنه مسألة أطول متتالية جزئية مشتركة. إن متتالية جزئية من متتالية معطاة هي المتتالية المعطاة نفسها وقد أُسْقِطَ منها صفر أو أكثر من عناصرها. فإذا كان لدينا المتتالية $X = \langle x_1, x_2, \ldots, x_m \rangle$، فإن متتالية أخرى $Z = \langle z_1, z_2, \ldots, z_k \rangle$ تكون ***متتالية جزئية subsequence*** من X إذا وُجدت متتالية متزايدة تمامًا $\langle i_1, i_2, \ldots, i_k \rangle$ من أدلة X بحيث أن لكل قيمة $j = 1, 2, \ldots, k$، يكون لدينا $x_{i_j} = z_j$. على سبيل المثال، إن $Z = \langle B, C, D, B \rangle$ هي متتالية جزئية من $X = \langle A, B, C, B, D, A, B \rangle$ والأدلة الموافقة هي $\langle 2, 3, 5, 7 \rangle$.

ليكن لدينا متتاليتان X و Y، نقول عن متتالية Z إنها ***متتالية جزئية مشتركة common subsequence*** لـ X و Y إذا كانت Z متتالية جزئية من كل من X و Y. مثلاً، إذا كانت $X = \langle A, B, C, B, D, A, B \rangle$ و $Y = \langle B, D, C, A, B, A \rangle$، فإن المتتالية $\langle B, C, A \rangle$ متتالية جزئية مشتركة بين X و Y. ولكن المتتالية $\langle B, C, A \rangle$ ليست *أطول* متتالية جزئية مشتركة Longest Common Subsequence (LCS)، فطول هذه المتتالية 3، وطول المتتالية $\langle B, C, B, A \rangle$، التي هي مشتركة أيضًا بين X و Y، هو 4. إن المتتالية $\langle B, C, B, A \rangle$ هي إحدى أطول المتتاليات المشتركة لـ X و Y، وكذلك الحال للمتتالية $\langle B, D, A, B \rangle$، إذ ليس لـ X و Y أي متتالية جزئية مشتركة بطول 5 أو أكثر.

في ***مسألة أطول متتالية جزئية مشتركة longest-common-subsequence problem (LCS)***، لدينا متتاليتان $X = \langle x_1, x_2, \ldots, x_m \rangle$ و $Y = \langle y_1, y_2, \ldots, y_n \rangle$ ونود إيجاد متتاليةٍ جزئيةٍ مشتركة بين X و Y بطول أعظمي. يبيّن هذا المقطع أنه يمكن حل هذه المسألة بفعالية باستخدام البرمجة الديناميكية.

الخطوة 1: توصيف أطول متتالية جزئية مشتركة

يعتمد نهج البحث الشامل في حل مسألة LCS على عَدِّ كلِّ المتتاليات الجزئية في X وفحص كلٍّ منها لنرى: هل هي متتالية جزئية من Y أيضًا؟ وتتبُّع أطول متتالية جزئية نجدها. توافِق كلُّ متتاليةٍ جزئية من X مجموعةً جزئيةً من الأدلة $\{1, 2, \ldots, m\}$ في X. ولما كان ثمة 2^m متتالية جزئية في X، فإن هذا الأسلوبَ يتطلب زمنًا أُسيًّا، ويجعله غير عملي للمتتاليات الطويلة.

ولكن للمسألة LCS خاصية البنية الجزئية المثلى، كما تبيّن المبرهنة التالية. وسنرى لاحقًا أن الصفوف الطبيعية للمسائل الجزئية توافق أزواجًا من "السوابق prefixes" لمتتاليتَي الدخل. ولكي نتوخّى الدقة، إذا كانت لدينا المتتالية $X = \langle x_1, x_2, \dots, x_m \rangle$، فإننا نُعرّف ***السابقة i prefix*** للمتتالية X لقيم $i = 0, 1, \dots, m$ على أنها: $X_i = \langle x_1, x_2, \dots, x_i \rangle$. فمثلاً، إذا كانت $X = \langle A, B, C, B, D, A, B \rangle$، فإن $X_4 = \langle A, B, C, B \rangle$ و X_0 هي المتتالية الخالية.

مبرهنة 15-1 (البنية الجزئية المثلى لمسألة LCS)

لتكن $X = \langle x_1, x_2, \dots, x_m \rangle$ و $Y = \langle y_1, y_2, \dots, y_n \rangle$ متتاليتان، ولتكن $Z = \langle z_1, z_2, \dots, z_k \rangle$ أي متتالية LCS لـ X و Y.

1. إذا كانت $x_m = y_n$، فإن $z_k = x_m = y_n$ و Z_{k-1} هي LCS لـ X_{m-1} و Y_{n-1}.
2. إذا كانت $x_m \neq y_n$، فإن $z_k \neq x_m$ تقتضي أن Z هي LCS لـ X_{m-1} و Y.
3. إذا كانت $x_m \neq y_n$، فإن $z_k \neq y_n$ تقتضي أن Z هي LCS لـ X و Y_{n-1}.

البرهان (1) إذا كان $z_k \neq x_m$ يمكننا أن نُلحِق $x_m = y_n$ بـ Z للحصول على متتالية جزئية مشتركة لـ X و Y بطول $k + 1$، وهذا يناقض افتراضَ أن Z أطولُ متتاليةٍ جزئية مشتركة لـ X و Y. لذلك يجب أن يكون $z_k = x_m = y_n$. ثم إن السابقة Z_{k-1} هي متتالية جزئية مشتركة بين X_{m-1} و Y_{n-1}، بطول $k - 1$. نود أن نثبت أنها LCS. افترض –بهدف نقض الفرض– أن ثمة متتالية جزئية مشتركة بين X_{m-1} و Y_{n-1} بطول أكبر من $k - 1$، ولتكن هذه المتتالية W. بعد ذلك، نُلحِق $x_m = y_n$ بـ W لإنتاج متتالية جزئية مشتركة بين X و Y، طولها أكبر من k، وهذا تناقض.

(2) إذا كان $z_k \neq x_m$ فإن Z متتالية جزئية مشتركة بين X_{m-1} و Y. ولو كان ثمة متتالية جزئية W مشتركة بين X_{m-1} و Y، طولها أكبر من k، لكانت W أيضًا متتالية جزئية مشتركة بين X_m و Y، وهذا يناقض فرضنا بأن Z هي LCS لـ X و Y.

(3) البرهان مماثل للحالة (2). ■

إن الطريقة التي تُوصّف بها المبرهنةُ 1.15 المتتالياتِ الجزئيةَ المشتركةَ ذات الطول الأعظم تدل على أن LCS لمتتاليتين تتضمن LCS لسابقتين للمتتاليتين. إذن فإن لمسألة LCS خاصيةَ البنية الجزئية المثلى. ويتّصف الحلُّ العودي بخاصية تراكب المسائل الجزئية، كما سنرى قريبًا.

الخطوة 2: حل عودي

نستنتج من المبرهنة 1.15 أن علينا دراسة مسألة جزئية واحدة أو اثنتين حين البحث عن LCS

i	x_i \ y_j	j=0	1 (B)	2 (D)	3 (C)	4 (A)	5 (B)	6 (A)
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

الشكل 8.15 الجدولان b و c تحسبان من LCS-LENGTH على المتتاليتين $X = \langle A, B, C, B, D, A, B \rangle$ و $Y = \langle B, D, C, A, B, A \rangle$. يحتوي المربع في السطر i والعمود j القيمة $c[i, j]$ والسهم المناسب للقيمة $b[i, j]$. العنصر 4 في $c[7, 6]$ —الزاوية اليمنى السفلى في الجدول– هو طول $\text{LCS}\langle B, C, B, A \rangle$ لـ X و Y. في حالة $i, j > 0$ يعتمد العنصر $c[i, j]$ فقط على كون $x_i = y_i$ وعلى القيم في العناصر $c[i-1, j]$ و $c[i, j-1]$ و $c[i-1, j-1]$ المحسوبة قبل $c[i, j]$. لإعادة بناء عناصر LCS، اتبع سهام $b[i, j]$ من الزاوية اليمنى السفلى؛ المسار مظلل. كل " ↖ " على المسار المظلَّل يوافق عنصرًا (مميزًا أو مضاء highlighted) يكون فيه $x_i = y_i$ عضوًا من LCS.

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2      return
3  if b[i, j] == " ↖ "
4      PRINT-LCS(b, X, i − 1, j − 1)
5      print x_i
6  elseif b[i, j] == " ↑ "
7      PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

يَطبع هذا الإجراءُ، للجدول b في الشكل 8.15، المتتاليةَ $BCBA$. يستغرق الإجراء زمنًا $O(m+n)$، لأنه في كل مرحلة من العودية يجري على الأقل إنقاص i أو j واحدًا.

تحسين الرماز

ما إن تستكمل تطوير خوارزميةً، حتى تجد في الأعم الأغلب أنك تستطيع تحسينها من جهة زمن التنفيذ أو فضاء الذاكرة الذي تستخدمه. يمكن لبعض التغييرات أن تبسط الرماز وتحسن عوامل ثابتة، غير أنها لا تؤدي إلى تحسينات الأداء المقارب. ويمكن أن تؤدي تغييرات أخرى إلى اختزالات مقاربة جوهرية في الزمن وفضاء الذاكرة.

ففي الخوارزمية LCS، على سبيل المثال، يمكننا حذف الجدول b بكامله. ويعتمد العنصر $c[i, j]$ على ثلاثة عناصر أخرى فقط للجدول c هي: $c[i-1, j-1]$ و $c[i-1, j]$ و $c[i, j-1]$. فإذا علمنا قيمة $c[i, j]$، أمكننا تحديد أي القيم الثلاث السابقة قد استُخدِمَت في حسابها، في زمن $O(1)$، دون تفحص الجدول b. وبذلك نستطيع إعادة بناء LCS في زمن $O(m+n)$ باستخدام إجرائية مشابهة لـ PRINT-LCS. (يُطلب إليك في التمرين 15.4-2 أن تعطي شبه الرماز لذلك.) ومع أننا نوفر فضاء ذاكرة $\Theta(mn)$ بهذه الطريقة، إلا أن متطلبات فضاء الذاكرة المساعدة لحساب LCS لا تنقص نقصانًا مقاربًا لأننا على أي حال، نحتاج إلى فضاء $\Theta(mn)$ للجدول c.

ومع ذلك، يمكننا خفض متطلبات الفضاء على نحو مقارب لـ LCS-LENGTH لأنها تحتاج فقط إلى سطرين من الجدول c في كل مرة: السطر الذي هو في قيد الحساب والسطر السابق. (في الحقيقة، وكما يُطلب إليك في التمرين 15.4-4 أن تبيّن أن بالإمكان استخدام فضاء أكبر بقليل من فضاء السطر الواحد من c لحساب طول LCS.) ويصحُّ هذا التحسين إذا كنا نحتاج إلى طول LCS فقط؛ أما إذا احتجنا إلى إعادة بناء عناصر الـLCS فإن الجدول الأصغر لا يحتفظ بالمعلومات الكافية لاقتفاء أثر خطواتنا في زمن $O(m+n)$.

تمارين

15.4-1

حدّد LCS لـ $\langle 1,0,0,1,0,1,0,1 \rangle$ و $\langle 0,1,0,1,1,0,1,1,0 \rangle$.

15.4-2

أعطِ شبه الرماز لإعادة بناء LCS من الجدول c الكامل والمتتاليتين الأصليتين $X = \langle x_1, x_2, \ldots, x_m \rangle$ و $Y = \langle y_1, y_2, \ldots, y_n \rangle$ في زمن $O(m+n)$، من دون استخدام الجدول b.

15.4-3

أعطِ نسخة مستذكَرة من LCS-LENGTH تُنفَّذ بزمن $O(mn)$.

15.4-4

بيّن طريقةَ حساب طول LCS باستخدام $2 \times \min(m, n)$ عنصرًا في الجدول c إضافة إلى $O(1)$ فضاء إضافي. ثم بيّن كيف يمكنك فعل ذلك باستخدام $\min(m, n)$ عنصرًا و $O(1)$ فضاء إضافي.

15.4-5

أعطِ خوارزمية تنفذ بزمن $O(n^2)$ لإيجاد أطول متتالية جزئية متزايدة باطراد من متتالية مؤلَّفةٍ من n عددًا.

★ *15.4-6*

أعطِ خوارزمية تنفذ بزمن $O(n \lg n)$ لإيجاد أطول متتالية جزئية متزايدة باطراد من متتالية مؤلَّفةٍ من n عددًا.

(*تلميح*: لاحظ أن آخر عنصر من متتالية جزئية مرشحة للإجابة بطول i هو على الأقل بكبر آخِر عنصر من متتالية جزئية مرشحة للإجابة بطول $i - 1$. احتفظ بالمتتاليات الجزئية المرشحة بربطها معًا من خلال متتالية الدخل.)

5.15 شجرات البحث الثنائية المثلى

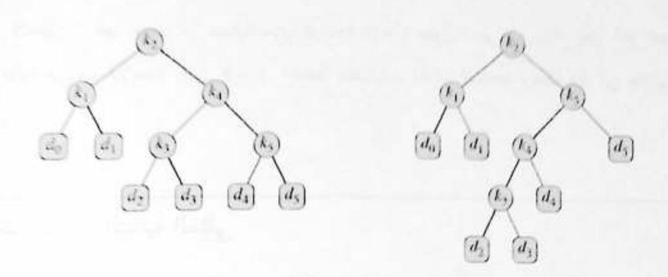
لنفترض أننا نصمم برنامجًا لترجمة نص من الإنكليزية إلى الفرنسية. إننا نحتاج، عند كلِّ ورود لكل كلمة إنكليزية في النص، إلى البحث عن مكافئها الفرنسي. يمكننا إنجاز عملية البحث هذه في بناء شجرة بحث ثنائية مفاتيحها n كلمة الإنكليزية، ومعطياتها التابعة هي المكافئات الفرنسية. ولأننا سنبحث في الشجرة عن كل كلمة مفردة في النص، نود أن يكون الزمن الكلي المصروف في البحث أصغر ما يمكن. يمكننا ضمان زمن بحثٍ $O(\lg n)$ لكل ورود باستخدام شجرة حمراء-سوداء أو أي شجرة بحث ثنائية متوازنة أخرى. على أن الكلمات تظهر بتواترات مختلفة، ويمكن أن تكون حالة الشجرة بحيث تظهر كلمة كثيرة الاستخدام مثل "*the*" بعيدة عن جذر الشجرة، في حين تَظهر كلمة نادرة الاستخدام مثل "*machicolation*" قريبة من الجذر. ومن شأن مثل هذا التنظيم أن يبطئ الترجمة، لأن عدد العُقد التي نزورها حين نبحث عن مفتاح في شجرة بحث ثنائية هو واحد مضافًا إلى عمق العقدة التي تتضمن المفتاح. ونحن نود أن توضع الكلمات العالية الورود في النص قرب الجذر.[6] أضف إلى ذلك إمكان ورود كلمات في النص ليس لها ترجمة فرنسية،[7] ويمكن ألا تظهر هذه الكلمات في شجرة البحث الثنائية على الإطلاق. فكيف يمكننا تنظيم شجرة البحث الثنائية بحيث يكون عدد العقد التي نزورها لجميع عمليات البحث أصغريًّا، إذا كنا نعلم تواتر ورود كل كلمة؟

ما نريده يُعرف باسم ***شجرة بحث ثنائية مثلى*** ***optimal binary search tree***. صوريًّا، لدينا متتالية $K = \langle k_1, k_2, \dots, k_n \rangle$ من n مفتاحًا متمايزًا، بترتيب مفروز (أي أن $k_1 < k_2 < \cdots < k_n$)، ونودّ بناء شجرة بحث ثنائية من هذه المفاتيح. لدينا لكل مفتاح k_i الاحتمال p_i وهو احتمال أن يكون البحث عن k_i. يمكن لبعض عمليات البحث أن تكون لقيم ليست في K، لذلك لدينا أيضًا $n + 1$ "مفتاحًا شكليًّا" هي $d_0, d_1, d_2, \dots, d_n$ تمثل قيمًا غير موجودة في K. وعلى وجه الخصوص، تمثل d_0 كل القيم التي هي أقل من k_1، وتمثل d_n كل القيم التي هي أكبر من k_n. وفي حالة $i = 1, 2, \dots, n - 1$، فإن المفتاح الشكلي d_i يمثل كل القيم التي هي بين k_i و k_{i+1}. ولكل مفتاح شكلي d_i لدينا الاحتمال q_i أن يتطابق البحث مع d_i. يبيّن الشكل 9.15 شجرتَي بحث ثنائيتين لمجموعة من خمس مفاتيح، $n = 5$. يكوّن كل مفتاح k_i

---

[6] إذا كان موضوع النص حول بنيان القلاع، قد نودّ أن تظهر كلمة "*machicolation*" قرب الجذر.

[7] نعم للكلمة *machicolation* مقابل فرنسي: *mâchioulis*.



الشكل 9.15 شجرتا بحث ثنائيتان لمجموعة من خمسة مفاتيح، $n = 5$، لها الاحتمالات التالية:

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

(أ) شجرة بحث ثنائية بتكلفة بحث متوقعة (وسطى) 2.80. **(ب)** شجرة بحث ثنائية بتكلفة بحث متوقعة 2.75. هذه الشجرة مثلى.

عقدةٌ داخلية، وكل مفتاح شكلي d_i ورقة. ويكون كل بحث إما ناجحًا (يُوجد مفتاحًا ما k_i) وإما غير ناجح (يُوجد مفتاحًا شكليًا ما d_i)، وبذلك يكون لدينا:

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 . \qquad (10.15)$$

ولما كان لدينا احتمالات بحث لكل مفتاح ولكل مفتاح شكلي، فيمكننا تحديد التكلفة المتوقعة لبحثٍ ما في شجرة بحث ثنائية معطاة T. لنفترض أن التكلفة الفعلية للبحث تساوي عددَ العُقد التي تفحصها، أي عمق العقدة التي نجدها بالبحث في T مضافًا له 1. عندها تكون التكلفة الموقعة للبحث في T هي:

$$\begin{aligned}\text{E[search cost in } T] &= \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \text{depth}_T(d_i) \cdot q_i \ , \qquad (11.15)\end{aligned}$$

حيث يشير depth_T إلى عمق العقدة في الشجرة T. والمساواة الأخيرة تنتج بالضرورة من المعادلة (10.15). وفي الشكل 15-9(أ)، يمكننا حساب تكلفة البحث عقدة بعقدة:

عقدة	عمقها	الاحتمال	المساهمة
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
المجموع			2.80

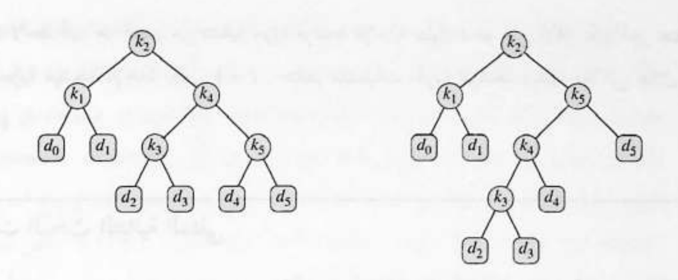
وفي حالة مجموعةٍ معطاةٍ من الاحتمالات، فإن هدفنا هو بناء شجرة بحث ثنائية بحيث تكون تكلفة البحث المتوقعة فيها أصغر ما يمكن. نسمي مثل هذه الشجرة ***شجرة بحث ثنائية مثلى*** ***optimal binary search tree***. ويبيّن الشكل 9.15(ب) شجرة بحث ثنائية مثلى للاحتمالات المعطاة في ترويسة الشكل؛ تكلفتها المتوقعة 2.75. يبيّن هذا المثال أن شجرة البحث الثنائية المثلى ليست بالضرورة الشجرة ذات الارتفاع الكلي الأصغر. وأنه لا يمكننا بالضرورة بناء شجرة البحث الثنائية المثلى بوضع المفتاح ذي الاحتمال الأعلى عند الجذر. فهنا يكون احتمال البحث عن المفتاح k_5 أعلى من احتمال البحث عن أي مفتاح آخر، ومع ذلك فإن جذر شجرة البحث الثنائية المثلى هو k_2. (إن أدنى تكلفة متوقعة لأي شجرة بحث ثنائية يكون k_5 عند جذرها هي 2.85.)

وكما هو الحال في جداء سلسلة المصفوفات، يخفق البحث الشامل لكل الاحتمالات في تحقيق خوارزمية فعالة. وبإمكاننا وضع لصيقة على عُقد أي شجرة ثنائية من n عقدة تتضمن المفاتيح $k_1, k_2, \ldots, k_n$ لبناء شجرة بحث ثنائية، ثم نضيف المفاتيح الشكلية كورقات. وقد رأينا في المسألة 12-4 أن عدد الشجرات الثنائية التي تتضمن n عقدة هو $\Omega(4^n/n^{3/2})$، وبذلك يكون علينا فحص عددٍ أُسّي من شجرات البحث الثنائية في البحث الشامل. وليس مستغربًا، أن نحلّ المسألة بالبرمجة الديناميكية.

الخطوة 1: بنية شجرة بحث ثنائية مثلى

لتوصيف البنية الجزئية المثلى لأشجار البحث الثنائية المثلى، نبدأ بملاحظة تتعلّق بالشجرات الفرعية. خُذْ أي شجرة فرعية من شجرة بحث ثنائية؛ يجب أن تتضمن هذه الشجرة مفاتيح في مجال متصل $k_i, \ldots, k_j$، لقيم $1 \le i \le j \le n$. إضافة إلى ذلك، فإن الشجرة الفرعية التي تتضمن المفاتيح $k_i, \ldots, k_j$، يجب أن تتضمن أيضًا المفاتيح الشكلية $d_{i-1}, \ldots, d_j$ باعتبارها وريقات لها.

الآن يمكننا التعبير عن البنية الجزئية المثلى: إذا كان لدينا شجرة بحث ثنائية مثلى T تحتوي شجرة فرعية



الشكل 9.15 شجرتا بحث ثنائيتان لمجموعة من خمسة مفاتيح، $n = 5$، لها الاحتمالات التالية:

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

(أ) شجرة بحث ثنائية بتكلفة بحث متوقعة (وسطى) 2.80. **(ب)** شجرة بحث ثنائية بتكلفة بحث متوقعة 2.75. هذه الشجرة مثلى.

عقدةٌ داخلية، وكل مفتاح شكلي d_i ورقة. ويكون كل بحث إما ناجحًا (يُوجد مفتاحًا ما k_i) وإما غير ناجح (يُوجد مفتاحًا شكليًا ما d_i)، وبذلك يكون لدينا:

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 . \tag{10.15}$$

ولما كان لدينا احتمالات بحث لكل مفتاح ولكل مفتاح شكلي، فيمكننا تحديد التكلفة المتوقعة لبحثٍ ما في شجرة بحث ثنائية معطاة T. لنفترض أن التكلفة الفعلية للبحث تساوي عددَ العُقد التي نفحصها، أي عمق العقدة التي نجدها بالبحث في T مضافًا له 1. عندها تكون التكلفة الموقعة للبحث في T هي:

$$\begin{aligned} \text{E[search cost in } T] &= \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \text{depth}_T(d_i) \cdot q_i \ , \end{aligned} \tag{11.15}$$

حيث يشير depth_T إلى عمق العقدة في الشجرة T. والمساواة الأخيرة تنتج بالضرورة من المعادلة (10.15). وفي الشكل 15-9(أ)، يمكننا حساب تكلفة البحث عقدة بعقدة:

عقدة	عمقها	الاحتمال	المساهمة
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
المجموع			2.80

وفي حالة مجموعةٍ معطاةٍ من الاحتمالات، فإن هدفنا هو بناء شجرة بحث ثنائية بحيث تكون تكلفة البحث المتوقعة فيها أصغر ما يمكن. نسمي مثل هذه الشجرة ***شجرة بحث ثنائية مثلى optimal binary search tree***. ويبيّن الشكل 9.15(ب) شجرة بحث ثنائية مثلى للاحتمالات المعطاة في ترويسة الشكل؛ تكلفتها المتوقعة 2.75. يبيّن هذا المثال أن شجرة البحث الثنائية المثلى ليست بالضرورة الشجرة ذات الارتفاع الكلي الأصغر. وأنه لا يمكننا بالضرورة بناء شجرة البحث الثنائية المثلى بوضع المفتاح ذي الاحتمال الأعلى عند الجذر. فهنا يكون احتمال البحث عن المفتاح k_5 أعلى من احتمال البحث عن أي مفتاح آخر، ومع ذلك فإن جذر شجرة البحث الثنائية المثلى هو k_2. (إن أدنى تكلفة متوقعة لأي شجرة بحث ثنائية يكون k_5 عند جذرها هي 2.85.)

وكما هو الحال في جداء سلسلة المصفوفات، يخفق البحث الشامل لكل الاحتمالات في تحقيق خوارزمية فعالة. وبإمكاننا وضع لصيقة على عُقد أي شجرة ثنائية من n عقدة تتضمن المفاتيح $k_1, k_2, \ldots, k_n$ لبناء شجرة بحث ثنائية، ثم نضيف المفاتيح الشكلية كورقات. وقد رأينا في المسألة 12-4 أن عدد الشجرات الثنائية التي تتضمن n عقدة هو $\Omega(4^n/n^{3/2})$، وبذلك يكون علينا فحص عددٍ أُسّي من شجرات البحث الثنائية في البحث الشامل. وليس مستغربًا، أن نحلّ المسألة بالبرمجة الديناميكية.

الخطوة 1: بنية شجرة بحث ثنائية مثلى

لتوصيف البنية الجزئية المثلى لأشجار البحث الثنائية المثلى، نبدأ بملاحظة تتعلّق بالشجرات الفرعية. خُذْ أي شجرة فرعية من شجرة بحث ثنائية؛ يجب أن تتضمن هذه الشجرة مفاتيح في مجال متصل $k_i, \ldots, k_j$، لقيم $1 \le i \le j \le n$. إضافة إلى ذلك، فإن الشجرة الفرعية التي تتضمن المفاتيح $k_i, \ldots, k_j$، يجب أن تتضمن أيضًا المفاتيح الشكلية $d_{i-1}, \ldots, d_j$ باعتبارها وريقات لها.

الآن يمكننا التعبير عن البنية الجزئية المثلى: إذا كان لدينا شجرة بحث ثنائية مثلى T تحتوي شجرة فرعية

T' تتضمن المفاتيح $k_i, ..., k_j$، وجب أن تكون الشجرة الفرعية T' مثلى أيضًا للمسائل الجزئية ذات المفاتيح $k_i, ..., k_j$ والمفاتيح الشكلية $d_{i-1}, ..., d_j$. وتصحّ هنا قاعدة "قص والصق" المعتادة. لو كان ثمة شجرة فرعية T'' تكلفتها المتوقعة أقل من التكلفة المتوقعة لـ T'، عندها يمكننا قصّ T' من T ولصق T'' مكانها، فتنتج شجرة بحث ثنائية بتكلفة متوقعة أقل من التكلفة المتوقعة لـ T، وهذا يناقض كون T مثلى.

نحتاج إلى استخدام بنية جزئية مثلى لنبيّن أننا نستطيع بناء حل أمثل للمسألة من الحلول المثلى للمسائل الجزئية. فإذا كان لدينا المفاتيح $k_i, ..., k_j$، فإن أحد هذه المفاتيح وليكن k_r حيث $(i \le r \le j)$، سيكون جذر شجرة فرعية مثلى تتضمن هذه المفاتيح. وتحتوي الشجرة الفرعية اليسرى للجذر k_r المفاتيحَ $k_i, ..., k_{r-1}$ (والمفاتيح الشكلية $d_{i-1}, ..., d_{r-1}$)، وتحتوي الشجرة الفرعية اليمنى المفاتيحَ $k_{r+1}, ..., k_j$ (والمفاتيح الشكلية $d_r, ..., d_j$). ومادمنا نفحص كل الجذور المرشحة k_r، حيث $i \le r \le j$، ونحدِّد كلَّ شجرات البحث الثنائية المثلى التي تتضمن المفاتيح $k_i, ..., k_{r-1}$ وتلك التي تتضمن المفاتيح $k_{r+1}, ..., k_j$، فمن المؤكد أننا سنجد شجرة بحث ثنائية مثلى.

ثمة تفصيلٌ جديرٌ بالملاحظة يتعلَّق بالشجرات الفرعية "الفارغة". لنفترض أننا، في شجرة فرعية تتضمن المفاتيح $k_i, ..., k_j$، اخترنا k_i جذرًا. فاستنادًا إلى الحجة المبينة آنفًا، فإن الشجرة الفرعية اليسرى التي جذرها k_i تتضمن المفاتيح $k_i, ..., k_{i-1}$. ومن الطبيعي أن نفسر هذه المتتالية على أنها لا تتضمن أية مفاتيح. ولكن، تذكَّر دائمًا أن الشجرات الفرعية تتضمن أيضًا مفاتيح شكلية. وسنعتمد اصطلاح أن الشجرة الفرعية التي تتضمن المفاتيح $k_i, ..., k_{i-1}$ لا تتضمن مفاتيح فعلية، لكنها تتضمن المفتاح الشكلي الوحيد d_{i-1} فقط. وبالتناظر، لو اخترنا k_j جذرًا، فإن الشجرة الفرعية اليمنى التي جذرها k_j تتضمن المفاتيح $k_{j+1}, ..., k_j$؛ ولا تتضمن هذه الشجرة الفرعية اليمنى أي مفاتيح فعلية، إلا أنها تتضمن المفتاح الشكلي d_j.

الخطوة 2: حل عودي

أصبحنا الآن مهيئين لتعريف قيمة الحل الأمثل عوديًّا. نتناول نطاقَ مسألتنا الجزئية على أنه إيجاد شجرة بحث ثنائية مثلى تتضمن المفاتيح $k_i, ..., k_j$، حيث $i \ge 1$ و $j \le n$ و $j \ge i - 1$. (لاحظ أنه عندما يكون لدينا $j = i - 1$ لا يكون ثمة أي مفاتيح فعلية؛ ويكون لدينا فقط المفتاح الشكلي d_{i-1}.) لنعرّف $e[i, j]$ على أنها التكلفة المتوقعة للبحث في شجرة بحث ثنائية مثلى تتضمن المفاتيح $k_i, ..., k_j$. ونود في نهاية المطاف حساب $e[1, n]$.

تحدث الحالة السهلة حين يكون $j = i - 1$. عندها يكون لدينا فقط المفتاح الشكلي d_{i-1}، وتكون تكلفة البحث المتوقعة $e[i, i-1] = q_{i-1}$.

وعندما يكون $j \ge i$، نحتاج إلى اختيار جذر k_r من بين المفاتيح $k_i, ..., k_j$ ثم إنشاء شجرة بحث ثنائية مثلى بحيث تكوّن المفاتيح $k_i, ..., k_{r-1}$ شجرتها الفرعية اليسرى وإنشاء شجرة بحث ثنائية مثلى بحيث تكوّن

المفاتيح $k_{r+1}, \dots, k_j$ شجرتها الفرعية اليمنى. ماذا يحصل لتكلفة البحث المتوقعة لشجرة فرعية حين تصبح هذه الشجرةُ الفرعية شجرةً فرعية لعقدة؟ يزداد عمق كل عقدة في الشجرة فرعية بـ 1. ومن العلاقة (11.15) تزداد تكلفةُ البحث المتوقعة لهذه الشجرة الفرعية بمجموع كل الاحتمالات في الشجرة الفرعية. وفي حالة شجرة فرعية تتضمن المفاتيح $k_i, \dots, k_j$، لنرمز إلى مجموع الاحتمالات هذا بـ

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \ . \tag{12.15}$$

ومن ثَمّ، إذا كان k_r جذر شجرةٍ فرعيةٍ مثلى تتضمن المفاتيح $k_i, \dots, k_j$ يكون لدينا

$$e[i,j] = p_r + \big(e[i,r-1]) + w(i,r-1) + \big(e[r+1,j] + w(r+1,j)\big) \ .$$

لاحظ أن:

$$w(i,j) = w(i,r-1) + p_r + w(r+1,j) \ ,$$

نعيد كتابة $e[i,j]$ كما يلي

$$e[i,j] = r[i,r-1] + e[r+1,j] + w(i,j) \tag{13.15}$$

تَفترض العلاقةُ العودية (13.15) أننا نعلم أي عقدة k_r سنختارها جذرًا. نختار الجذر الذي يعطي أقل تكلفة بحث متوقعة، وهذا يعطينا الصيغة التكرارية النهائية:

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \ , \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j \ . \end{cases} \tag{14.15}$$

تعطي قيمُ $e[i, j]$ تكاليفَ البحث المتوقعة في شجرات البحث الثنائية المثلى. ولمساعدتنا في اقتفاء أثر بنية شجرات البحث الثنائية المثلى، نُعرّف $root[i, j]$ للقيم $1 \le i \le j \le n$ على أنه الدليل r حيث k_r جذر شجرة بحث ثنائية مثلى تتضمن المفاتيح $k_i, \dots, k_j$. ومع أننا سنرى طريقةَ حساب قيم $root[i, j]$، سندع بناء شجرة البحث الثنائية المثلى من هذه القيم للتمرين 15.5-1.

الخطوة 3: حساب تكلفة البحث المتوقعة لشجرة بحث ثنائية مثلى

لعلك لاحظت، عند هذه النقطة، التشابهات بين توصيفنا لشجرات البحث الثنائية المثلى وجداء سلسلة المصفوفات. ففي كلا نطاقَي المسألتين، تتكون مسائلنا الجزئية من مجالات جزئية ذات أدلة متصلة. وسيكون التنجيز المباشر العودي للمعادلة (14.15) غير فعال كما كان الحال في الخوارزمية المباشرة العودية لجداء سلسلة مصفوفات. وعوضًا عن ذلك، نخزن قيم $e[i,j]$ في جدول $e[1 \,..\, n+1, 0 \,..\, n]$. يحتاج الدليل الأول أن يذهب حتى $n+1$ عوضًا عن n، ذلك أننا لكي نحصل على شجرة فرعية تتضمن المفتاح الشكلي d_n فقط، نحتاج إلى حساب $e[n+1,n]$ وخزنها. ويحتاج الدليل الثاني أن يبدأ من 0 كي نحصل على شجرة فرعية

تتضمن المفتاح الشكلي d_0 فقط، لذلك نحتاج إلى حساب $e[1,0]$ وخزنها. نستخدم العناصر $e[i,j]$ التي يكون لها $j \geq i-1$ فقط. ونستخدم أيضًا الجدول $root[i,j]$ لتسجيل جذر الشجرة الفرعية التي تتضمن المفاتيح $k_i, \dots, k_j$. يستخدم هذا الجدول العناصر التي تحقق $1 \leq i \leq j \leq n$ فقط.

سنحتاج إلى جدول آخر لزيادة الفعالية. فعوضًا عن حساب $w(i,j)$ من العدم في كل مرة نحسب فيها $e[i,j]$ – وهذا يستغرق $\Theta(j-i)$ عملية جمع– فإننا نخزن هذه القيم في جدول $w[1\,..\,n+1, 0\,..\,n]$. وفي الحالة الأساسية، نحسب $w[i,i-1] = q_{i-1}$ حين يكون $1 \leq i \leq n+1$. أما في حالة $j \geq i$ فإننا نحسب:

$$w[i,j] = w[i,j-1] + p_j + q_j \ . \tag{15.15}$$

وهكذا يمكننا حساب $\Theta(n^2)$ قيمةً لـ $w[i,j]$ بزمن $\Theta(1)$ لكل منها.

إن دخل شبه الرماز التالي هو: الاحتمالات $p_1, \dots, p_n$ و $q_0, \dots, q_n$ والحجم n. وهو يعيد الجدولين e و $root$.

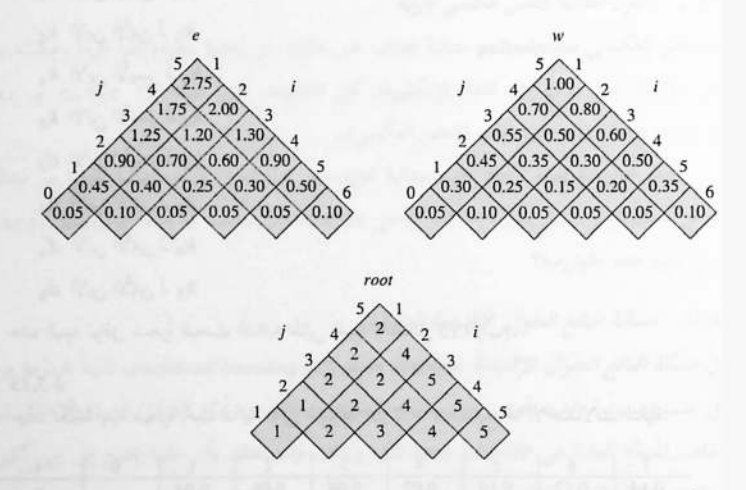
```
OPTIMAL-BST(p, q, n)
 1  let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
 2  for i = 1 to n + 1
 3      e[i, i − 1] = q_{i−1}
 4      w[i, i − 1] = q_{i−1}
 5  for l = 1 to n
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          e[i, j] = ∞
 9          w[i, j] = w[i, j − 1] + p_j + q_j
10          for r = i to j
11              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
12              if t < e[i, j]
13                  e[i, j] = t
14                  root[i, j] = r
15  return e and root
```

من هذا التوصيف، ومن التشابه مع إجرائية MATRIX-CHAIN-ORDER في المقطع 2.15، ستجد أن عمل هذه الإجرائية واضح ومباشر. فحلقة **for** في السطور 2-4 تستبدئ قيم $e[i, i-1]$ و $w[i,i-1]$. ثم تَستخدم حلقةُ **for** في السطور 5-14 العلاقاتِ العودية (14.15) و (15.15) لحساب $e[i,j]$ و $w[i,j]$ للقيم $1 \leq i \leq j \leq n$. وفي التكرار الأول حين يكون $l = 1$ تَحسب الحلقةُ $e[i,i]$ و $w[i,i]$ لكل $i = 1,2,\dots,n$. وفي التكرار الثاني، حين تكون $l = 2$، يجري حساب $e[i,i+1]$ و $w[i,i+1]$ لكل

$i = 1,2, \ldots, n-1$، وهكذا. أما حلقة **for** الأعمق (الأكثر داخلية) في السطور 10-14 فتجرِّب كل دليل مرشح r لتحدد أي مفتاح k_r يجب استخدامه جذرًا لشجرة بحث ثنائية مثلى تتضمن المفاتيح $k_i, \ldots, k_j$. تخزِّن حلقةُ **for** هذه القيمَ الحالية للدليل r في $root[i, j]$ كلما وجدت مفتاحًا أفضل يمكن استخدامه جذرًا.

يبيِّن الشكل 10.15 الجداول $e[i, j]$ و $w[i, j]$ و $root[i, j]$ المحسوبة في الإجرائية OPTIMAL-BST لتوزيع المفاتيح المبيّن بالشكل 9.15. وكما هو الحال في مثال جداء سلسلة المصفوفات للشكل 5.15، تدوَّر الجداول لجعل الأقطار أفقية. تحسب OPTIMAL-BST السطورَ من الأسفل إلى الأعلى ومن اليسار إلى اليمين ضمن كل سطر.

تستغرق الإجرائية OPTIMAL-BST زمنًا $\Theta(n^3)$، تمامًا مثل MATRIX-CHAIN-ORDER. من السهل ملاحظة أن زمن التنفيذ $O(n^3)$، لأن الإجرائية تضم ثلاث حلقات **for** متداخلة، وكل دليل حلقة يأخذ قيمًا تساوي n على الأكثر. وليس لأدلة الحلقات في OPTIMAL-BST الحدود نفسها تمامًا الموجودة في MATRIX-CHAIN-ORDER، إلا أنها تختلف على الأكثر بـ 1 في كل الاتجاهات. لذلك، وكما في MATRIX-CHAIN-ORDER، تستغرق الإجرائيةُ OPTIMAL-BST زمنًا $\Omega(n^3)$.



الشكل 10.15 الجداول $e[i, j]$ و $w[i, j]$ و $root[i, j]$ محسوبةً بالإجرائية OPTIMAL-BST لتوزيع المفاتيح المبيّن بالشكل 9.15. جرى تدوير الجداول لتبدو الأقطار أفقية.

تمارين

1-5.15

اكتب شبه رماز للإجرائية CONSTRUCT-OPTIMAL-BST(*root*) التي تُخرِج بنيةَ شجرة بحث ثنائية مثلى، من جدول *root* معطى. يجب أن تطبع الإجرائيةُ البنيةَ التالية في حالة مثال الشكل 10.15.

k_2 is the root
k_1 is the left child of k_2
d_0 is the left child of k_1
d_1 is the right child of k_1
k_5 is the right child of k_2
k_4 is the left child of k_5
k_3 is the left child of k_4
d_2 is the left child of k_3
d_3 is the right child of k_3
d_4 is the right child of k_4
d_5 is the right child of k_5

والتي تعني على الترتيب:

k_2 هي الجذر
k_1 الابن الأيسر لـ k_2
d_0 الابن الأيسر لـ k_1
d_1 الابن الأيمن لـ k_1
k_5 الابن الأيمن لـ k_2
k_4 الابن الأيسر لـ k_5
k_3 الابن الأيسر لـ k_4
d_2 الابن الأيسر لـ k_3
d_3 الابن الأيمن لـ k_3
d_4 الابن الأيمن لـ k_4
d_5 الابن الأيمن لـ k_5

هذه البنية توافق شجرة البحث الثنائية المثلى المبينة بالشكل 9.15(ب).

2-5.15

حدّد تكلفة بنية شجرة بحث ثنائية مثلى لمجموعة من $n = 7$ مفاتيح لها الاحتمالات التالية:

7	6	5	4	3	2	1	0	i
0.14	0.12	0.10	0.02	0.08	0.06	0.04		p_i
0.05	0.05	0.05	0.05	0.06	0.06	0.06	0.06	q_i

3-5.15

افترض أنه عوضًا عن الاحتفاظ بالجدول $w[i, j]$، فإننا حسبنا القيمة $w(i, j)$ مباشرة من المعادلة (12.15) في السطر 9 من OPTIMAL-BST واستخدمنا هذه القيمة المحسوبة في السطر 11. كيف يمكن أن يؤثر هذا التغيير في زمن التنفيذ المقارب لـ OPTIMAL-BST؟

★ *4-5.15*

بيَّن كنوث Knuth في المرجع [212] أنه يوجد دائمًا جذور لشجرات فرعية مثلى بحيث $root[i, j-1] \le root[i, j] \le root[i+1, j]$ لكل القيم $1 \le i < j \le n$. استخدم هذه الحقيقة لتعديل الإجرائية OPTIMAL-BST كي تُنفَّذ بزمن $\Theta(n^2)$.

مسائل

1-15 *أطول مسار بسيط في بيان موجه غير دوّار*

افترض أن لدينا بيانًا موجَّهًا غيرَ دوار $G = (V, E)$ مع قيم حقيقية لأوزان الوصلات، وعقدتين متباينتين s و t. وصِّف نهجًا بالبرمجة الديناميكية لإيجاد أطول مسار بسيط مثقَّل من s إلى t. ما شكل بيان المسألة الجزئية؟ وما هي فعالية خوارزميتك؟

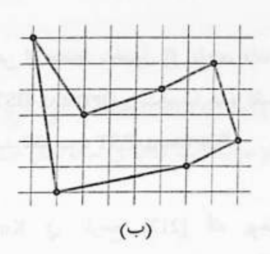
2-15 *أطول متتالية جناس عكسي جزئية*

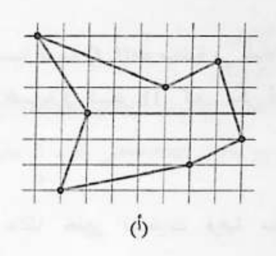
الجناسُ العكسي palindrome متتاليةُ محارف غير خالية، من أبجدية معينة، تُقرأ طردًا وعكسًا دون تغيير. من الأمثلة على ذلك في اللغة الإنكليزية، كل المتتاليات التي طولها 1: `civic` و `racecar` و `aibohphobia` (أي رهاب الجناس العكسي).

أعطِ خوارزمية فعالة لإيجاد أطول متتالية محارف من هذا النوع، تكون متتالية جزئية من متتالية محارف دخْلٍ معطاة. على سبيل المثال، إذا كان الدخل `character`، يجب أن تعيد خوارزميتُك `carac`. ما هو زمن تنفيذ هذه الخوارزمية؟

3-15 *مسألة البائع الجوال الإقليدية البيتونية*

في ***مسألة البائع الجوال الإقليدية euclidean traveling-salesman problem***، لدينا مجموعة من n نقطة في المستوي، ونودّ تحديد أصغر جولة مغلقة تربط جميع نقاط n. يبيّن الشكل 11.15(أ) الحل لمسألة بـ 7 نقاط. المسألة العامة هي NP-صعبة NP-hard، ولذلك فإننا نعتقد بأن حلها يحتاج إلى زمن أكبر من كثير حدودي (انظر الفصل 34).





الشكل 11.15 سبع نقاط في المستوي، مبيَّنة على شبكة واحدية. (أ) أقصر جولة مغلقة، بطول تقريبي 24.89. هذه الجولة ليست بيتونية. (**ب**) أقصر جولة بيتونية لمجموعة النقاط نفسها. طولها تقريبًا 25.58.

يرى J.L. Bentley أن نبسِّط المسألة بقصر اهتمامنا على ***الجولات البيتونية bitonic tours***، أي الجولات التي تبدأ من النقطة في أقصى اليسار وتذهب حصرًا من اليسار إلى اليمين إلى النقطة في أقصى اليمين، ثم تعود حصرًا من اليمين إلى اليسار إلى نقطة البدء. يبيّن الشكل 15-11(ب) أقصر جولة بيتونية للنقاط الـ 7 نفسها. في هذه الحالة، يمكن إيجاد خوارزمية بزمن كثير حدودي.

وصِّفْ خوارزميةً بزمن $O(n^2)$ لتحديد جولة بيتونية مثلى. يمكن أن تفترض أنه لا يوجد أي نقطتين لهما إحداثيات x (الفاصلة) نفسها. (*تلميح*: أجرِ المسحَ من اليسار إلى اليمين، محتفظًا بإمكانات مثلى للجزأين في الجولة.)

15-4 الطباعة المتقنة

لنأخذ بالاعتبار مسألة طباعة فقرةٍ بإتقان بنمط متساوي الفراغات (لجميع المحارف العرض نفسه) بطابعة. النص المُدخل هو متتالية من n كلمة بأطوال $l_1, l_2, \ldots, l_n$ مَقيسَة بالمحارف. نود طباعة هذه الفقرة بإتقان على عدد من السطور يتسع كل منها لـ M محرفًا على الأكثر. معيارنا "للإتقان" هو كما يلي: إذا تضمن سطر ما الكلمات من i إلى j، بحيث $i \le j$ وتركنا فراغًا واحدًا بالضبط بين الكلمات، فإن عدد محارف الفراغ الإضافية في آخر السطر هو $M - j + i - \sum_{k=i}^{j} l_k$، الذي يجب أن يكون غير سالب بحيث يتسع السطر للكلمات. نود تصغير مجموع مكعبات عدد الفراغات الإضافية في أواخر السطور، على كل الأسطر ما عدا السطر الأخير. أعطِ خوارزمية برمجة ديناميكية تَطبع فقرةً من n كلمة بإتقان على طابعة. حلّل متطلبات زمن التنفيذ وفضاء الذاكرة لخوارزميتك.

15-5 مسافة التحرير

لكي نحول سلسلة محارف مصدرية من نص $x[1..m]$ إلى سلسلة محارف وجهة $y[1..n]$، يمكننا تطبيق عدة عمليات تحويل. وغرضنا، إذا كانت لدينا x و y، أن ننتج سلسلة التحويلات التي تحوّل x إلى y.

نستخدم صفيفة z، نفترضها كبيرة كفاية لتتسع كلَّ المحارف اللازمة، لنضع فيها النتائج المتوسطة. بدايةً تكون z فارغة، وفي النهاية، يجب أن يكون لدينا $z[j] = y[j]$ لقيم $j = 1, 2, \dots, n$. نحتفظ بالأدلة الحالية i على x والأدلة j على z، ويُسمح للعمليات أن تغير z وهذه الأدلة. بداية، $i = j = 1$، والمطلوب فحص كل محرف في x خلال عملية التحويل، وهذا يعني أنه في نهاية متتالية عمليات التحويل، يجب أن يكون لدينا $i = m + 1$.

يمكننا الاختيار من بين ست عمليات تحويل:

نسخ محرفٍ من x إلى z بوضع $z[j] = x[i]$، ثم إضافة واحد إلى كلٍّ من i و j. هذه العملية تفحص $x[i]$.

استعاضة عن محرفٍ من x بمحرف آخر c بوضع $z[j] = c$، ثم إضافة واحد إلى كلٍّ من i و j. هذه العملية تفحص $x[i]$.

حذف محرف من x بزيادة واحد على i، وإبقاء j على حالها. هذه العملية تفحص $x[i]$.

إدراج محرف c في z بوضع $z[j] = c$، ثم زيادة واحد على j، وإبقاء i على حالها. هذه العملية لا تفحص أيًّا من محارف x.

فتل (أي مبادلة) المحرفين التاليين بنسخهما من x إلى z، ولكن بالترتيب المعاكس؛ نفعل ذلك كما يلي: $z[j] = x[i+1]$ و $z[j+1] = x[i]$، ثم بوضع $i = i + 2$ و $j = j + 2$. هذه العملية تفحص $x[i]$ و $x[i+1]$.

قتل ما تبقى من x بوضع $i = m + 1$. تفحص هذه العملية كل محارف x التي لم تُفحَص بعد. إذا نُفذَت هذه العملية كانت بالضرورة العملية النهائية.

كمثال على ذلك، فإن من بين طرق تحويل سلسلة المحارف المصدرية `algorithm` إلى سلسلة المحارف الوجهة `altruistic` هي استخدام المتتالية التالية من العمليات، حيث المحارف التي تحتها خط هي $x[i]$ و $z[j]$ بعد العملية:

العملية	x	z
سلسلة المحارف الأولية	`algorithm` (a مسطر)	–
نسخ	`algorithm` (l مسطر)	`a_`
نسخ	`algorithm` (g مسطر)	`al_`
استعاضة عن المحرف بـ t	`algorithm` (o مسطر)	`alt_`
حذف	`algorithm` (r مسطر)	`alt_`
نسخ	`algorithm` (i مسطر)	`altr_`

altru_	algorithm	إدراج u
altrui_	algorithm	إدراج i
altruis_	algorithm	إدراج s
altruisti_	algorithm	فتل (مبادلة)
altruistic_	algorithm	إدراج c
altruistic_	algorithm_	قتل

لاحظ أنه توجد عدة متتاليات أخرى لعمليات تحويل تحوِّل algorithm إلى altruistic.

إن لكل عملية تحويل تكلفة مرفقة بها. وتعتمد تكلفةُ عملية ما على خصوصية التطبيق، إلا أننا نفترض أن تكلفة كل عملية هي مقدار ثابت معلوم لنا. نفترض أيضًا أن التكاليف الفردية لعمليات النسخ والاستعاضة أقل من مجموع تكلفة عمليتي الحذف والإدراج؛ وإلا فإننا لا نستخدم عمليات النسخ والاستعاضة. إن تكلفة متتالية ما من عمليات التحويل تساوي مجموع تكاليف العمليات الفردية في المتتالية. وفي حالة المتتالية السابقة تكون تكلفة تحويل algorithm إلى altruistic هي:

(3 × تكلفة (نسخ)) + تكلفة (استعاضة) + تكلفة (حذف) + (4 × تكلفة (إدراج))
+ تكلفة (مبادلة) + تكلفة (قتل).

أ. إذا كانت لدينا متتاليتان $x[1..m]$ و $y[1..n]$ ومجموعة من عمليات التحويل مع تكاليفها، فإن ***مسافة التحرير edit distance*** من x إلى y هي تكلفة متتالية العمليات الأرخص ثمنًا التي تحول x إلى y. وَصِّفْ خوارزميةَ برمجةٍ ديناميكية توجد مسافةَ التحرير من $x[1..m]$ إلى $y[1..n]$ وتَطبع متتالية عمليات مثلى. حلِّل متطلبات زمن التنفيذ وفضاء الذاكرة لخوارزميتك.

إن مسألة مسافة التحرير تعميمٌ لمسألة رصف متتاليتَي DNA (انظر على سبيل المثال، المقطع 2.3 في المرجع Setubal و Meidanis [310]). ثمة عدة طرق لقياس التشابه بين متتاليتي DNA برصفهما. تتألف إحدى طرق صفّ متتاليتين x و y من إدراج فراغات في مواضع اعتباطية (لا على التعيين) في المتتاليتين (ومنها مواضع عند إحدى النهايات) بحيث يكون للمتتاليات الناتجة x' و y' الطول نفسه، ولا يكون هناك فراغ في الموضع نفسه (أي لا يوجد أي موضع j بحيث يكون كلٌّ من $x'[j]$ و $y'[j]$ فراغًا). ثم نعيِّن "علامة تقييم" لكل موضع. يستقبل الموضع j العلامة كما يلي:

- $+1$ إذا كان $x'[j] = y'[j]$ وليس فيهما فراغ.
- -1 إذا كان $x'[j] \neq y'[j]$ وليس فيهما فراغ.
- -2 إذا كان $x'[j]$ أو $y'[j]$ فراغًا.

علامة عملية الرصف هي مجموع علامات العمليات المفردة. على سبيل المثال، إذا كان لدينا المتتاليتان $x =$ GATCGGCAT و $y =$ CAATGTGAATC فإن إحدى عمليات الرصف هي:

```
G ATCG GCAT
CAAT GTGAATC
-*++*+*+-++*
```

يشير + تحت موضع ما إلى علامة تساوي 1+ لذلك الموضع، ويشير − إلى علامة 1- و * إلى علامة 2-، بحيث يكون لعملية الرصف هذه علامةٌ إجماليةٌ $6 \cdot 1 - 2 \cdot 1 - 4 \cdot 2 = -4$.

ب. اشرح طريقةَ صوغ مسألة إيجاد عملية رصفٍ مثلى باعتبارها مسألة مسافة تحرير باستخدام مجموعة جزئية من عمليات التحويل: نسخ واستعاضة وحذف وإدراج ومبادلة وقتل.

6-15 التخطيط لحفلة شركة

يعمل الأستاذ ستيوارت Stewart مستشارًا لمدير شركة تخطط لحفلة في الشركة. وللشركة بنية هرمية (تراتبية)؛ أي إن علاقات المشرفين تؤلِّف شجرةً جذرها الرئيس. وقد أسند مكتب الموظفين إلى كل موظف ترتيبًا بحسب درجة مرحه، يمثل عددًا حقيقيًّا. ولجعل الحفلة متعةً لكل الحضور، لا يريد الرئيس أن يحضر الموظف مع مسؤوله المباشر.

أُعطي الأستاذ ستيوارت الشجرة التي تُوصِّف بنية الشركة باستخدام التمثيل: الابن الأيسر والأخ الأيمن المُوصَّف في المقطع 4.10. وتحمل كلُّ عقدةٍ من الشجرة، إضافة إلى المؤشرات، اسمَ الموظف وترتيبه في المرح. وصِّف خوارزميةً لتأليف قائمة الضيوف، بحيث يكون مجموع تقديرات الضيوف المرحة أعظميًّا. حلِّل زمن تنفيذ خوارزميتك.

7-15 خوارزمية فيتربي

يمكننا استخدام البرمجة الديناميكية على بيانٍ موجَّه $G = (V, E)$ لتعرُّف الكلام. توضع على كل وصلة $(u, v) \in E$ لصيقة بصوت $\sigma(u, v)$ من مجموعة منتهية Σ من الأصوات. إن البيان مع لصيقاته هو نموذج صوري لشخص يتكلم لغة محددة. يبدأ كلُّ مسارٍ في البيان من عقدةٍ مميزة $v_0 \in V$ توافق متتاليةً ممكنةً من الأصوات التي ينتجها النموذج. تُعرَّف لصيقة مسار موجَّه بأنها تتابع للصيقات الوصلات على المسار.

أ. وصِّف خوارزمية فعالة تستطيع – بوجود بيانٍ G على وصلاته لصيقات، وعقدة مميزة v_0 ومتتالية من المحارف $s = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ من Σ – أن تعيد مسارًا في G يبدأ من v_0 ولصيقته s، في حال وجود مثل هذا المسار. وإلا فيجب أن تعيد الخوارزمية عبارة NO-SUCH-PATH. حلِّل زمن تنفيذ خوارزميتك. (*تلميح*: يمكن أن تجد بعض المفاهيم المفيدة في الفصل 22.)

الآن افترض أننا أعطينا كل وصلة $(u, v) \in E$ احتمالاً غير سالب $p(u, v)$ لاجتياز الوصلة (u, v) من

العقدة u، منتجة بذلك الصوت الموافق. إن مجموع احتمالات الوصلات التي تنطلق من أي عقدة يساوي 1. ويعرَّف احتمال المسار بأنه جداء احتمالات وصلاته. ويمكننا النظر إلى احتمال مسار يبدأ من v_0 على أنه احتمال "سَيْر عشوائي" يبدأ من v_0 ويتبع المسار المحدد، حيث نختار عشوائيًا الوصلةَ المطلوبة، وندع العقدة u وفقًا لاحتمالات الوصلات المتاحة التي تغادر u.

ب. وسع إجابتك على الجزء (أ) بحيث إذا أعيد مسارٌ ما فهو المسار الأكثر احتمالاً الذي يبدأ من v_0 ولصيقته s. حلِّل زمن تنفيذ خوارزميتك.

8-15 ضغط الصورة بنقش عروق

لتكن لدينا صورة ملونة تتكون من صفيفة $A[1..m, 1..n]$، أبعادها $m \times n$، من البكسلات pixels (أو العنصورات)، حيث يُحدِّد كل بكسل ثلاثيةً من كثافات الألوان: الأحمر والأخضر والأزرق (RGB). افترض أننا نودّ ضغط هذه الصورة ضغطًا بسيطًا. وبالتحديد، نريد أن نحذف بكسلاً من كل سطر من الأسطر m، بحيث تصبح الصورة بكاملها أضيق ببكسل واحد. ولكن، لتجنّب آثار التشوه البصري، نتطلب أن يقع البكسلان، اللذان نحذفهما من سطرين متجاورين، في العمود نفسه أو في عمودين متجاورين؛ تشكل البكسلات المحذوفة "عِرقًا أو درزة" من أعلى سطر إلى أدنى سطر حيث تكون البكسلات المتتالية في العِرق متجاورة شاقوليًا أو قطريًا.

أ. بيّن أن عدد مثل هذه العروق الممكنة يزداد أُسيًّا في m على الأقل بافتراض أن $n > 1$.

ب. افترض الآن أننا حسبنا مع كل بكسل $A[i,j]$ قياسَ تمزيق $d[i,j]$ قيمتُه حقيقية، مشيرةً إلى مدى التمزّق الحاصل من حذف البكسل $A[i,j]$. ويلاحَظ بالبديهة أنه كلما كان قياس تمزق البكسلات أصغر، كان البكسل أكثر شبهًا بمجاوراته. افترض أيضًا أننا عرّفنا قياس تمزّق العِرق على أنه مجموع تمزّقات بكسلاته.

أعطِ خوارزميةً لإيجاد العِرق ذي قياس التمزق الأصغر. ما مدى فعالية خوارزميتك؟

9-15 قطع متتاليات محرفية

تتيح بعض لغات معالجة المتتاليات المحرفية للمبرمج قَطْعَ متتاليةٍ محرفيةٍ إلى قطعتين. ولأن هذه العملية تَنسخ المتتاليةَ المحرفية، فإنها تكلف n وحدة زمن لقطع متتالية من n محرفًا إلى قطعتين. افترض أن المبرمج يريد قطع المتتالية إلى عدة قطع؛ إن الترتيب الذي يحصل وفقه القطع يمكن أن يؤثر في الزمن الكلي المستغرَق. على سبيل المثال، افترض أن المبرمجَ يريد أن تُقطع متتاليةٌ من 20 محرفًا بعد المحارف 2 و 8 و 10 (بترقيم المحارف بالترتيب المتصاعد من اليسار، بدءًا من الرقم 1). فإذا بَرْمَجَ القَطْعَ بحيث يحدث بالترتيب من اليسار إلى اليمين، فإن القطع الأول يكلف 20 وحدة زمن، والثاني يكلف 18 وحدة زمن (قطع المتتالية من المحرف 3 إلى

المحرف 20 عند المحرف 8). ويكلف القطع الثالث 12 وحدة زمن، ويكون المجموع 50 وحدة زمن. أما إذا بَرْمَجَ القَطْعَ بحيث يحدث بالترتيب من اليمين إلى اليسار، فعندها سيكلف القطع الأول 20 وحدة زمن، والثاني 10 وحدات زمن، والثالث 8 وحدات زمن، ويكون المجموع 38 وحدة زمن. وفي ترتيب آخر مختلف، يمكن أن يَقطع المبرمجُ عند المحرف 8 أولاً (بتكلفة 20)، ثم يَقطع القطعة اليسرى عند المحرف 2 (بتكلفة 8)، وأخيرًا القطعة اليمنى عند المحرف 10 (بتكلفة 12)، ويكون إجمالي التكلفة 40.

صمِّمْ خوارزميةً، إذا أُعطِيَت عددًا من المحارف يجري بعدها القَطْع، تُحدِّدُ الطريقةَ ذات التكلفة الدنيا لترتيب عمليات القطع هذه. وعلى نحو صوري أكثر، إذا أعطيت متتالية محرفية S، مؤلَّفةً من n محرفًا، وصفيفة $L[1..m]$ تتضمن نقاط القطع، احسب أدنى تكلفة لمتتالية القطع، مع متتالية القطع التي تُنجَز بهذه التكلفة.

15-10 تخطيط استراتيجية استثمار

تساعدك معرفتك بالخوارزميات على الحصول على عمل مثير في شركة Acme Computer Company مع مكافأة 10,000 دولار عند التوقيع. قرَّرتَ استثمار هذه الأموال بهدف جعل دخلك أعظميًّا في نهاية 10 سنوات. وقرَّرتَ استخدام شركة Amalagamated Investment Company لإدارة استثماراتك. تَطلب هذه الشركةُ احترامَ القواعد التالية: إنها تقدم n استثمارًا مختلفًا، مرقمة من 1 إلى n. وفي كل عام j، يزوِّدك الاستثمار i بمعدل دخل (ربح) r_{ij}. وبتعبير آخر: إذا استثمرت d دولارًا بالاستثمار رقم i من العام j، فإنك تحصل على dr_{ij} في نهاية العام j. ومعدلات الدخل مضمونة، أي إنك تحصل على كل معدلات الدخل على مدى السنوات العشر التالية لكل استثمار. ويمكنك اتخاذ قرارات استثمار مرة واحدة في السنة. فإذا قرّرتَ أن تترك أموالك في المجموعة نفسها من الاستثمار عامَيْن متتالين، عليك أن تسدد رسمًا مقداره f_1 دولارًا، أما إذا قررتَ تحويل أموالك إلى مجموعة استثمار مختلفة فإنك تسدد رسمًا مقداره f_2 دولارًا، حيث $f_2 > f_1$.

أ. تسمح لك المسألة، حسبما ذُكرت، باستثمار أموالك في عدة استثمارات كل سنة. أثبتْ أنه توجد استراتيجية مثلى، تكمن في وضع جميع الأموال في استثمارٍ واحدٍ في كلِّ سنة. (تذكَّر أن استراتيجية الاستثمار المثلى تجعل المبلغ بعد 10 سنوات أعظميًّا، وهي غير معنيَّةٍ بأيِّ أغراضٍ أخرى، كتقليص الأخطار إلى الحدود الدنيا.)

ب. برهن أن مسألة تخطيط استراتيجية الاستثمار المثلى تكشف عن بنيةٍ جزئيةٍ مثلى.

ت. صمِّمْ خوارزميةً تخطِّط استراتيجيةَ استثمارك المثلى. ما زمن تنفيذ خوارزميتك؟

ث. افترض أن شركة Amalgamated Investments فرضت قيودًا إضافية بحيث لا يمكنك، في كل لحظة،

وضع أكثر من 15,000 دولارٍ في استثمارٍ واحد. بَيِّنْ أن مسألة تعظيم دخلك بعد 10 سنوات لم تعد تُظهِر بنيةً جزئيةً مثلى.

15-11 تخطيط المخزون

تُنتج شركةُ Rinky Dink Company آلاتٍ لتجديد سطوح المزالج الجليدية. يتغير الطلب على مثل هذه المنتجات من شهر لآخر، ولذلك تحتاج الشركة إلى تطوير استراتيجية لتخطيط تصنيعها في ضوء الطلب المتقلِّب والمتوقَّع في الوقت نفسه. ترغب الشركة في تصميم خطة للأشهر الـ n التالية. ولكل شهر i تعلم الشركةُ عددَ الطلبات d_i أي عدد الآلات التي ستبيعها. ليكن $D = \sum_{i=1}^{n} d_i$ مجموع الطلبات على مدى الأشهر n التالية. تحتفظ الشركة بموظفيها الذين يعملون بدوام كامل للقيام بالعمل اللازم لتصنيع m آلة في الشهر. إذا احتاجت الشركة إلى تصنيع أكثر من m آلة في شهر معين، يمكنها استئجار عمال بدوام جزئي، بتكلفة تبلغ c دولارًا للآلة الواحدة. وإضافةً إلى ذلك، إذا بقي لدى الشركة، في نهاية الشهر، أي آلات لم تُبَعْ، فإنها تسدد تكاليف تخزينها. تُعطى تكلفة الاحتفاظ بـ j آلة كدالة $h(j)$ للقيم $j = 1, 2, \ldots, D$ حيث $h(j) \geq 0$ في حالة $1 \leq j \leq D$ ولدينا أيضًا $h(j) \geq h(j+1)$ في حالة $1 \leq j \leq D\text{-}1$.

أعطِ خوارزمية لحساب خطة للشركة تخفِّض التكاليفَ إلى حدودها الدنيا في الوقت الذي تحقِّق فيه جميع الطلبات. يجب أن يكون زمنُ التنفيذ كثير حدود في n و D.

15-12 التعاقد مع لاعبي البيسبول الأحرار

افترض أنك مدير عام لفريق بيسبول من الفئة الأولى. ربما تحتاج، خارج الموسم، إلى التعاقد مع لاعبين أحرار لفريقك. ولقد أعطاك مالك الفريق ميزانية X دولارًا للإنفاق على اللاعبين الأحرار. وسمح لك أن تنفق أقل من X دولارًا بالمجموع، ولكنه سيستغني عن خدماتك إذا أنتَ تجاوزت في الإنفاق X دولارًا.

لديك N موقعًا مختلفًا، ولكل موقع يوجد P لاعبًا حرًّا متاحًا ليلعب ذلك الموقع.[8] ولما كنتَ لا تريد أن تثقل قائمتك بعددٍ كبيرٍ من اللاعبين لموقعٍ ما، فبإمكانك أن تتعاقد مع لاعبٍ واحدٍ على الأكثر لكلِّ موقع. (إذا لم تتعاقد مع أيِّ لاعبٍ لموقع معين، فعليك أن تُبقي على لاعبيك لذلك الموقع.)

ولكي تحدد قيمة لاعبٍ ما، فإنك تقرر استخدام إحصائيات saber[9] تُعرَف بـ "VORP" أو "value over replacement player" أي "القيمة عند تبديل لاعب". فلاعبٌ ذو قيمة VORP عالية أغلى ثمنًا من

---

[8] مع أنه يوجد تسعة مواقع لفريق البيسبول، فإن N لا تساوي بالضرورة 9 لأن لبعض المديرين طرقًا خاصة للتفكير في المواقع. فمثلاً، قد يعتبر المدير أن للرماة اليمينين والرماة اليساريين مواقع منفصلة، وكذلك رماة الاستهلال ورماة الاحتياط الذين يمكنهم الرماية في عدة جولات، ورماة الاحتياط الذين يرمون عادة في جولة واحدة على الأكثر.

[9] هو تطبيق التحليل الإحصائي على سجلات البيسبول. وهو يتيح عدة طرق لمقارنة القيم النسبية لأفراد اللاعبين.

لاعبٍ ذي VORP منخفضة. إلا أن اللاعب ذا VORP عالية ليس بالضرورة أغلى ثمنًا للتعاقد من لاعب ذي VORP منخفضة، إذ إن ثمة عوامل أخرى غير ثمن اللاعب تحدد تكلفة التعاقد معه.

تؤخذ في الاعتبار ثلاثةُ أمور بشأن كلِّ لاعبٍ موجود:

- موقع اللاعب
- تكلفة تعاقد اللاعب
- قيمة VORP للاعب.

صمِّمْ خوارزميةً تجعل مجموع VORP للاعبين الذين تتعاقد معهم أعظميًا، على ألاَّ يتجاوز مجموع إنفاقك X دولارًا. يمكنك أن تفترض أن تَعاقُدَ كلِّ لاعب من مضاعفات 100,000 دولار. ويجب أن يكون خرجُ خوارزميتك: مجموع قيمة VORP للاعبين الذين تتعاقد معهم، ومجموع المال الذي تنفقه، وقائمة باللاعبين الذين وقع اختيارك عليهم للتعاقد. حلِّل متطلبات زمن التنفيذ والذاكرة المطلوبة لخوارزميتك.

ملاحظات الفصل

استهلَّ R. Bellman الدراسةَ المنهجية للبرمجة الديناميكية في عام 1955. وتشير كلمة "برمجة" – هنا وفي البرمجة الخطية معًا – إلى استخدام طريقة الحل المُجَدوَل. ومع أن تقنيات الأمثَلَة التي تتضمن عناصر البرمجة الديناميكية كانت معروفة من قبل، فقد زوَّد بيلمان هذا المجال بأساسٍ رياضي متين [37].

وصنف Galil و Park [125] خوارزميات البرمجة الديناميكية بحسب حجم الجدول وعدد عناصر الجدول الأخرى التي يعتمد عليها كلُّ عنصر. وهما يسمِّيان خوارزمية البرمجة الديناميكية tD/eD إذا كان حجم جدولها $O(n^t)$ وكان كلُّ عنصرٍ يعتمد على $O(n^e)$ عنصرًا آخر. فمثلاً، خوارزمية جداء المصفوفات الواردة في المقطع 2.15 هي $2D/1D$، وخوارزمية أطول متتالية محرفية مشتركة الواردة في المقطع 4.15 هي $2D/0D$.

وأعطى Hu و Shing [182, 183] خوارزميةً بزمن $O(n \lg n)$ لمسألة جداء سلسلة مصفوفات.

ويبدو أن الخوارزمية التي تنفذ بزمن $O(mn)$ لمسألة إيجاد أطول متتالية جزئية مشتركة هي خوارزمية شعبية. فقد طرح Knuth [71] تساؤلاً عن إمكان وجود خوارزميات بزمن أقل من الرتبة التربيعية (من الدرجة الثانية) subquadratic algorithms لمسائل LCS. وأجاب Masek و Paterson [244] عن هذا السؤال بالإيجاب، وذلك بإعطاء خوارزمية تُنفَّذ بزمن $O(mn/\lg n)$، حيث $n \le m$ والمتتاليات مأخوذة من مجموعة محدودة الحجم. وفي الحالة الخاصة، التي لا يَظهر فيها أيُّ عنصرٍ أكثر من مرة واحدة في متتالية الدخل، بيَّن Szymanski [326] كيف يمكن حل المسألة بزمن $O((n+m)\lg(n+m))$. وينطبق كثيرٌ من هذه النتائج على مسألة حساب مسافات تحرير سلسلة محارف (المسألة 15-5).

ثمة مقالة قديمة كتبها Gilbert و Moore [133] عن الترميزات الاثنانية المتغيرة الطول، كان لها تطبيقات في بناء شجرات بحث ثنائية مثلى للحالة التي تكون فيها قيم جميع الاحتمالات p_i تساوي 0؛ تضمنت هذه المقالةُ خوارزميةً بزمن $O(n^3)$. وكذلك قدَّم Aho وHopcroft و Ullman [5] الخوارزميةَ الواردةَ في المقطع 5.15. ويعود التمرين 5.15-4 إلى Knuth [212]. على حين ابتكر Hu وTucker [184] خوارزميةً للحالة التي تكون فيها الاحتمالات p_i مساويةً للصفر تَستخدم زمنًا $O(n^2)$ وفضاءً $O(n)$؛ وفيما بعد قَلَّص كنوث [211] الزمن إلى $O(n \lg n)$.

وتعود المسألة 15-8 إلى Avidan و Shamir [27]، اللذين وضعا فيديو رائعًا على الوب يبيّن هذه التقنية في ضغط الصورة.

16 الخوارزميات الشرهة

تَمُرُّ خوارزميات مسائل الأمثَلَة عادةً بمتتالية من الخطوات، مع مجموعة من الخيارات عند كل خطوة. وفي كثيرٍ من مسائل الأمثَلَة، يُعَدُّ استخدام البرمجة الديناميكية لتحديد أفضل الخيارات إفراطًا؛ على أنَّ ثمة خوارزمياتٍ أبسطَ وأكثر فاعلية يمكن أن تفيَ بالغرض. تختار ***الخوارزميةُ الشرهة greedy algorithm*** دائمًا الخيارَ الذي يبدو أنه الأفضل في تلك اللحظة. أي إنها تأخذ الخيار الأفضل محليًّا، على أمل أن يقود هذا الخيار إلى حلٍّ أمثلَ شامل. يستكشف هذا الفصل مسائلَ الأمثَلة التي يمكن حلُّها بخوارزميات شرهة. وقبل قراءة هذا الفصل، ينبغي قراءة ما ورد عن البرمجة الديناميكية في الفصل 15، وبوجهٍ خاص المقطع 3.15.

إن الخوارزميات الشرهة لا تتوصل دائمًا إلى حلولٍ مُثْلَى، لكنها تتوصل إليها في الكثير من المسائل. بدايةً، سندرس في المقطع 1.16، مسألة بسيطة إلا أنها غير تافهة: وهي مسألة اختيار النشاطات. وهي مسألة تَحسب لها خوارزميةٌ شرهةٌ حلاً أمثل بفعالية. سنصل إلى الخوارزمية الشرهة باعتماد طريقة البرمجة الديناميكية أولاً، ثم نبيِّن أن بإمكاننا دائمًا القيام بخيارات شرهة للوصول إلى حلٍّ أمثل. يستعرض المقطعُ 2.16 العناصرَ الأساسية للنهج الشره، معطيًا نهجًا أبسط لبرهان صحة الخوارزميات الشرهة. ويقدم المقطع 3.16 تطبيقًا هامًا للتقنيات الشرهة: تصميم أرمزة هوفمان Huffman لضغط المعطيات. ونتفحَّص في المقطع 4.16 جزءًا نظريًّا، يؤسس للبنى التوافقية المسماة "كيانات مصفوفية matroid"، التي تُنتِج الخوارزميةُ الشرهةُ دائمًا حلاً أمثل لها. أخيرًا، يبيِّن المقطع 5.16 تطبيق الكيانات المصفوفية باستخدام مسألة جدولة مهام في واحدة الزمن مع مدد انتهاء وعقوبات.

إن الطريقة الشرهة قوية وتعمل جيدًا على مجال واسع من المسائل. ستقدم الفصول اللاحقة عدة خوارزميات يمكن رؤيتها على أنها تطبيقات للطريقة الشرهة، ومنها خوارزميات شجرة المسح الصغرى (الفصل 23)، وخوارزمية Dijkstra لأقصر الطرق من مصدر وحيد (الفصل 24)، وكسبية Chvátal الشرهة لتغطية مجموعة (الفصل 35). ومع أنه يمكن قراءة هذا الفصل والفصل 23 على نحو مستقل، ولكن قد تجد أن قراءتهما معًا مفيدة.

1.16 مسألة اختيار النشاطات

مثالنا الأول هو مسألة جدولة عدة نشاطات متنافسة تتطلب استخدام مورد مشترك استخدامًا حصريًّا، والهدف هو اختيار مجموعة نشاطات متوافقة فيما بينها وذات حجم أعظم. افترض أن لدينا مجموعة من n ***نشاطًا activities*** $S = \{a_1, a_2, \ldots, a_n\}$ تريد جميعُها استخدامَ موردٍ ما (قاعة محاضرات مثلاً)، يمكن أن يُخدَّم نشاطًا واحدًا في كل مرة. لكل نشاطٍ a_i ***لحظة بداية start time*** s_i ***ولحظة انتهاء finish time*** f_i، حيث $0 \le s_i < f_i < \infty$. إذا جرى اختيار النشاط a_i فإنه يحدث خلال المجال الزمني نصف المفتوح $[s_i, f_i)$. نقول عن نشاطين a_i و a_j إنهما ***متوافقان compatibles*** إذا لم يكن المجالان $[s_i, f_i)$ و$[s_j, f_j)$ متراكبين overlap؛ أي إن النشاطين a_i و a_j متوافقان إذا كان $s_i \ge f_j$ أو $s_j \ge f_i$). إن ***مسألة اختيار النشاطات activity-selection problem*** هي مسألة اختيار مجموعة جزئية ذات عدد أعظم من النشاطات المتوافقة فيما بينها. لندرس، على سبيل المثال، المجموعةَ التالية S من النشاطات، التي جرى فرزها بحسب الترتيب المتزايد باطراد للحظات الانتهاء.

$$f_1 \le f_2 \le f_3 \le \cdots \le f_{n-1} \le f_n \, . \tag{1.16}$$

(سنرى لاحقًا الفائدة التي يحقِّقها هذا الافتراض.) لندرس مثلاً مجموعة النشاطات التالية S:

11	10	9	8	7	6	5	4	3	2	1	i
12	2	8	8	6	5	3	5	0	3	1	s_i
16	14	12	11	10	9	9	7	6	5	4	f_i

في هذا المثال، تتألف المجموعة $\{a_3, a_9, a_{11}\}$ من نشاطات متوافقة فيما بينها، ولكنها ليست مجموعة جزئية عُظْمَى، لأن المجموعة $\{a_1, a_4, a_8, a_{11}\}$ أكبر منها. والواقع أن المجموعة $\{a_1, a_4, a_8, a_{11}\}$ هي مجموعةٌ جزئيةٌ عُظْمَى من النشاطات المتوافقة؛ وثمة مجموعةٌ عُظْمَى أخرى هي $\{a_2, a_4, a_9, a_{11}\}$.

سنحل هذه المسألة بعدة خطوات؛ فنبدأ بالتفكير في حلِّ هذه المسألة بالبرمجة الديناميكية. في هذا الحل، نأخذ بالحسبان عدةَ خيارات حين نحدِّد المسائلَ الجزئية التي يجب استخدامها في الحل الأمثل. سنلاحظ لاحقًا أن علينا أن نأخذ بالحسبان خيارًا واحدًا – وهو الخيار الشره – وأننا حين نعتمده، فإن مسألة جزئية واحدة تبقى. اعتمادًا على هذه الملاحظات فإننا سنطوِّر خوارزميةً عَوْدِيةً شرهةً لحلِّ مسألة جدولة النشاطات. وسنتمِّم إجرائيةَ تطوير الحل الشره بتحويل الخوارزمية العَوْدية إلى خوارزميةٍ تكرارية. ومع أن الخطوات التي سنسير وفْقها في هذه المقطع هي أكثر تفصيلاً مما هو معتاد عند تطوير خوارزمية شرهة، إلا إنها تبيِّن العلاقةَ بين الخوارزميات الشرهة والبرمجة الديناميكية.

البنية الجزئية المُثْلَى في مسألة اختيار النشاطات

يمكننا التحقق بسهولة من أن مسألة اختيار النشاطات تُظهِر بنيةً جزئيةً مُثْلَى. لنرمز بـ S_{ij} إلى مجموعة النشاطات التي تبدأ قبل أن ينتهي النشاط a_i، وتنتهي قبل بدء النشاط a_j. ولنفترض أننا نودّ إيجادَ مجموعةٍ عُظْمَى من النشاطات المتوافقة فيما بينها في S_{ij}، ولنفترض أيضًا أن هذه المجموعة العُظْمَى هي A_{ij}، وأنها تتضمن نشاطًا ما a_k. فإذا ضمَّنا a_k في حلٍّ أمثلَ، فإننا أمام مسألتين جزئيتين: إيجادِ النشاطات المتوافقة فيما بينها في S_{ik} (النشاطات التي تبدأ بعد انتهاء النشاط a_i وتنتهي قبل بدء النشاط a_k)، وإيجادِ النشاطات المتوافقة فيما بينها في S_{kj} (النشاطات التي تبدأ بعد انتهاء النشاط a_k وتنتهي قبل بدء النشاط a_j). ولنكن $A_{ik} = A_{ij} \cap S_{ik}$ و $A_{kj} = A_{ij} \cap S_{kj}$، بحيث تتضمن A_{ik} النشاطات في A_{ij} التي تنتهي قبل بدء a_k، وتتضمن A_{kj} النشاطات في A_{ij} التي تبدأ بعد انتهاء a_k. وبذلك يكون لدينا $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$، وتتكوَّن المجموعةُ A_{ij} (ذاتُ الحجم الأعظم من النشاطات المتوافقة فيما بينها في S_{ij}) من $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$ نشاطًا.

تُظهِر حجةُ القص واللصق الاعتيادية وجوبَ أن يتضمَّن الحلُ الأمثل A_{ij} حلولاً مُثْلَى لكلٍّ من المسألتين الجزئيتين S_{ik} و S_{kj}. فلو كان بإمكاننا إيجاد مجموعةٍ A'_{kj} من النشاطات المتوافقة فيما بينها في S_{kj} حيث $|A'_{kj}| > |A_{kj}|$ لكان بوسعنا استخدام A'_{kj} عوضًا عن A_{kj} في حل المسألة الجزئية S_{ij}. وبذلك نكون قد بنينا مجموعةً من النشاطات المتوافقة فيما بينها $|A_{ik}| + |A'_{kj}| + 1 > |A_{ik}| + |A_{kj}| + 1 = |A_{ij}|$، وهذا يناقض كون A_{ij} حلاً أمثلَ. وتطبَّق حجةُ النظير على النشاطات في S_{ik}.

توحي هذه الطريقةُ في وصف بنية الحل الأمثل إلى أن بإمكاننا حلُّ مسألة اختيار النشاطات بالبرمجة الديناميكية. فإذا رمزنا إلى حجم الحلِّ الأمثل للمجموعة S_{ij} بـ $c[i,j]$، فيمكننا كتابة التكرار

$$c[i,j] = c[i,k] + c[k,j] + 1 \ .$$

بالطبع، إذا لم نكن نعلم أن حلاً أمثلَ للمجموعة S_{ij} يتضمن النشاط a_k لوجب علينا فحص جميع النشاطات في S_{ij} لإيجاد نشاط نختاره، بحيث:

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \ , \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \ . \end{cases} \qquad (2.16)$$

يمكننا بعدها تطوير خوارزمية عَوْدِية لاستذكار هذا الحل، أو يمكننا العمل صعوديًّا لملء عناصر الجدول مع تقدم الحل. ولكن في هذه الحالة نكون قد تجاوزنا خاصيةً هامةً أخرى لمسألة اختيار النشاطات، يمكننا أن نستفيد منها كثيرًا.

القيام بالخيار الشره

ماذا لو كان بإمكاننا اختيار نشاط لإضافته إلى حلنا الأمثل دون أن نكون ملزمين بحلّ جميع المسائل الجزئية

سلفًا؟ إن ذلك سيجنِّبنا دراسةَ جميع الخيارات الموجودة في التكرار (2.16). في الحقيقة، نحتاج في مسألة اختيار النشاطات إلى دراسةِ خيارٍ وحيدٍ هو: الخيار الشره.

ماذا نعني بالخيار الشره في مسألة اختيار النشاطات؟ يقترح الحدس أن نختار نشاطًا يترك المورد متاحًا لأكبر عددٍ ممكن من النشاطات الأخرى. والآن، يجب أن يكون أول النشاطاتِ انتهاءً أحد النشاطات التي نختارها. وهذا يقتضي أن نختار من S نشاطًا الأُولى انتهاءً، لأنه سيترك المورد متاحًا لأكبر عددٍ ممكن من النشاطات الأُخَر. (إذا وُجد أكثر من نشاط في S له الانتهاء الأبكر، أمكننا اختيار أيٍّ منها.) وبعبارةٍ أخرى، لما كانت النشاطات مفروزة بحسب ترتيب لحظات انتهائها المتزايدة باطِّراد، فإن الخيار الشره هو النشاط a_1. على أنَّ اختيارَ النشاط الذي ينتهي أولاً ليس الطريقةَ الوحيدةَ للقيام بالخيار الشره لهذه المسألة. يُطلب إليك في التمرين 1.16-3 استكشاف إمكانات أخرى.

إذا قمنا بالخيار الشره، يبقى علينا حلُّ مسألة جزئية واحدة فقط: إيجاد النشاطات التي تبدأ بعد انتهاء a_1. لماذا لا يجب علينا إيجاد النشاطات التي تنتهي قبل بدء a_1؟ لدينا $s_1 < f_1$ و f_1 هو الأبكر انتهاءً من أي نشاط، لذلك، لا يمكن لأي نشاط أن تكون لحظة انتهائه أقل أو تساوي s_1. وهكذا، فإن جميع النشاطات المتوافقة مع النشاط a_1 يجب أن تبدأ بعد انتهاء a_1.

يُضاف إلى ذلك، أننا برهنا سابقًا أن مسألة اختيار النشاطات تُظهِر بنيةً جزئيةً مُثْلَى. لتكن $S_k = \{a_i \in S : s_i \geq f_k\}$ مجموعة النشاطات التي تبدأ بعد انتهاء a_k. إذا قمنا بالاختيار الشره للنشاط a_1، فإن S_1 تبقى المسألة الجزئية الوحيدة التي علينا حلها.[1] نستنبط من البنى الجزئية المُثْلَى أنه إذا كان a_1 في الحل الأمثل، فإن الحلَّ الأمثل للمسألة الأصلية يتكوَّن من النشاط a_1 ومن كل النشاطات في حل أمثل للمسألة الجزئية S_1.

يبقى سؤال كبير واحد: هل حدسنا صحيح؟ هل الخيار الشره – الذي نختار فيه النشاط الذي ينتهي أولاً – هو دائمًا جزء من حلٍّ أمثلَ ما؟ تبيّن المبرهنة التالية أنه كذلك.

مبرهنة 1.16

لتكن S_k أية مسألة جزئية غير خالية، وليكن a_m نشاطًا في S_k له أبكر لحظة انتهاء، عندها يكون a_m مُضمَّنًا في مجموعة جزئية ذات حجم أعظم من نشاطات S_k المتوافقة فيما بينها.

البرهان لتكن A_k مجموعة جزئية ذات حجم أعظم من النشاطات المتوافقة فيما بينها في S_k. وليكن a_j النشاط في A_k ذا لحظة الانتهاء الأبكر. إذا كان $a_j = a_m$، فقد تحقَّق المطلوب، لأننا بيّنا أن a_m هي في مجموعة جزئية ذات حجم أعظم من النشاطات المتوافقة فيما بينها من S_k. أما إذا كان $a_j \neq a_m$، فنفترض

---

[1] نشير أحيانًا إلى المجموعات S_k على أنها مسائل جزئية بدلاً من كونها مجموعات نشاطات. وسيتضح دائمًا من السياق إذا كنا نشير بـ S_k إلى مجموعات النشاطات أو إلى مسائل جزئية مداخلها هذه المجموعات.

أن المجموعة $A'_k = A_k - \{a_j\} \cup \{a_m\}$ هي A_k مع الاستعاضة عن a_m بـ a_j. فتكون النشاطات في A'_k منفصلة، وذلك لأن النشاطات في A_k منفصلة، ويكون a_j هو أول نشاط ينتهي في A_k، و $f_m \leq f_j$. ولما كان $|A'_k| = |A_k|$، فإن A'_k هي مجموعة جزئية ذات حجم أعظم من النشاطات المتوافقة فيما بينها من S_k، وهي تتضمن a_m. ■

وهكذا، نرى أنه على الرغم من أننا قد نكون قادرين على حل مسألة اختيار النشاطات باستخدام البرمجة الديناميكية، إلا أننا لسنا ملزمين بها. (إضافة إلى أننا لم نختبر بعد إذا كان لمسألة اختيار النشاطات مسائل جزئية متراكبة.) عوضًا عن ذلك، يمكننا اختيار النشاط الذي ينتهي أولاً تكراريًّا، والإبقاء فقط على النشاطات المتوافقة معه، ونكرر ذلك حتى انتهاء النشاطات. يضاف إلى ذلك، أنه بسبب اختيارنا النشاطَ ذا لحظة الانتهاء الأبكر دائمًا، فيجب أن تكون لحظات انتهاء النشاطات التي نختارها متزايدة تمامًا. يمكننا إذن أن ندرس إمكان أخذ كل نشاط مرة واحدة خلال عملنا، وذلك تبعًا للترتيب المتزايد باطراد للحظات انتهاء النشاطات.

لا تحتاج الخوارزمية التي تحل مسألة اختيار النشاطات إلى أن تعمل صعوديًّا، كما هو الحال في خوارزمية البرمجة الديناميكية المعتمدة على الجداول. عوضًا عن ذلك، يمكنها أن تعمل نزوليًّا، باختيار نشاط ووضعه في الحل الأمثل، ثم بحل المسألة الجزئية المتمثلة باختيار النشاطات من بين تلك المتوافقة مع النشاطات المُختارَة سابقًا. للخوارزميات الشرهة عادةً هذا التصميم النزولي: حدِّد الخيار ثم حلَّ مسألة جزئية، وذلك عوضًا عن التقنية الصعودية القائمة على حلِّ المسائل الجزئية قبل تحديد الخيار.

خوارزميةٌ عَوْدِيةٌ شرهة

الآن بعد أن عرفنا كيف نتجاوز طريقةَ البرمجة الديناميكية، وبديلاً عن استخدام خوارزمية شرهة نزولية، يمكننا كتابةُ إجراءٍ عَوْدِي مباشر لحل مسألة اختيار النشاطات. يأخذ الإجراء RECURSIVE-ACTIVITY-SELECTOR لحظات بدء النشاطات ولحظات انتهائها، ممثَّلةً في صفيفتين s و f،[2] والمؤشر k الذي يعرِّف المسألة الجزئية S_k الواجب حلّها، و n حجم المسألة الأصلية. تعيد هذه الخوارزميةُ مجموعةً ذات حجم أعظم من النشاطات المتوافقة فيما بينها في S_k. نفترض أن نشاطات الدخل التي عددها n مرتبة بحسب لحظات الانتهاء المتزايدة باطراد تبعًا للمعادلة (1.16)، وإلا فيمكننا فرزها بهذا الترتيب خلال زمن $O(n \lg n)$، باختيار ترتيبٍ عشوائي في حالة المساواة. في البدء، نضيف نشاطًا وهميًّا a_0 حيث $f_0 = 0$، بحيث تكون المسألة الجزئية S_0 هي كامل مجموعة النشاطات S. الاستدعاء الابتدائي الذي يحل كامل المسألة هو RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$.

---

[2] لما كان شبه الرماز يعتبر s و f صفيفتين، فإنه يفهرس ضمنهما باستخدام أقواس مربعة عوضًا عن أدلة.

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]     // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

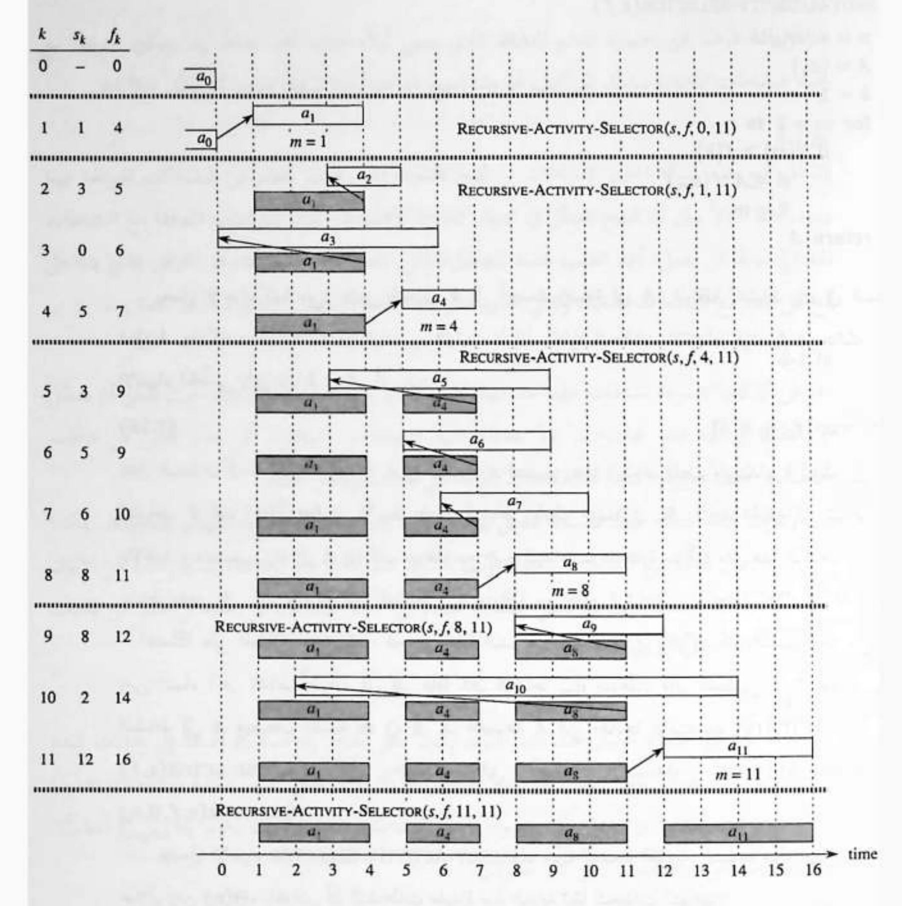
يبين الشكل 1.16 عمل هذه الخوارزمية. في أحد الاستدعاءات RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)، تبحث حلقة **while** في السطرين 2-3 عن أول نشاط ينتهي في S_k. تفحص الحلقة $a_{k+1}, a_{k+2}, \ldots, a_n$ إلى أن تجد أول نشاط a_m متوافق مع a_k؛ وهو نشاط يحقق $s_m \geq f_k$. إذا انتهت الحلقة – بسبب عثورها على مثل هذا النشاط – تعيد الإجرائيةُ في السطر 5 اجتماعَ $\{a_m\}$ والمجموعة الجزئية ذات الحجم الأعظم لـ S_m التي أعادها الاستدعاء RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n). وبالمقابل، يمكن أن تنتهي الحلقة لأن $m > n$، وفي تلك الحالة نكون قد فحصنا كل النشاطات في S_k دون أن نجد النشاط المتوافق مع a_k. وفي هذه الحالة، يكون $S_k = \emptyset$، وبذلك تعيد الإجرائية $\emptyset$ في السطر 6.

بافتراض أن النشاطات كانت مفروزة تصاعديًّا بحسب لحظات الانتهاء، يكون زمن تنفيذ الإجراء RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$ $\Theta(n)$، ويمكن أن نرى ذلك كما يلي. عبر كل الاستدعاءات العَوْدية، يجري فحص كل نشاط مرة واحدة تمامًا في اختبار حلقة **while** في السطر 2. وبوجهٍ خاص، يجري فحص النشاط a_i في آخر استدعاء كان فيه $k < i$.

خوارزميةٌ تكراريةٌ شرهة

يمكننا بسهولة تحويل إجرائنا العَوْدي إلى إجراء تكراري. إن الإجراء RECURSIVE-ACTIVITY-SELECTOR هو "عَوْدِيّ الذيل tail recursive" تقريبًا (انظر المسألة 7-4): فهو ينتهي باستدعاء عَوْدِي له نفسه يتلوه عملية اجتماع. إن مهمة تحويل إجراء عَوْدِيّ الذيل إلى الشكل التكراري هي عادة مهمة مباشرة. في الحقيقة، تنجز بعضُ مترجماتِ لغات البرمجة هذه المهمة آليًّا. يعمل الإجراء RECURSIVE-ACTIVITY-SELECTOR، كما هو مكتوب، على المسائل الجزئية S_k، أي إن المسائل الجزئية تتكون من النشاطات التي تنتهي آخرًا (آخر النشاطات من حيث الانتهاء).

إن الإجراء GREEDY-ACTIVITY-SELECTOR هو نسخة تكرارية من الإجراء RECURSIVE-ACTIVITY-SELECTOR. وهو يفترض أيضًا أن نشاطات الدخل مرتبة بحسب لحظات الانتهاء المتزايدة باطراد. فهو يجمع النشاطات المختارة في مجموعة A، ويعيد هذه المجموعة حين ينتهي.



الشكل 1.16 عمل الإجراء RECURSIVE-ACTIVITY-SELECTOR على 11 نشاطًا معطًى سابقًا. تَظهر النشاطاتُ المدروسةُ لكل استدعاء بين الخطوط الأفقية. ينتهي النشاط الوهمي a_0 في اللحظة 0، وفي الاستدعاء الأول RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, 11)$، يجري اختيار النشاط a_1. نرى، لكل استدعاء عَوْدِي، النشاطات التي جرى اختيارها سابقًا مظللة والنشاطات التي هي في قيد الدراسة بالأبيض. إذا كانت لحظة البدء لنشاط ما قبل لحظة انتهاء آخر نشاط مُضاف (السهم بينها يشير إلى اليسار)، يُستَبعَد هذا النشاط. وإلا (يشير السهم مباشرة إلى الأعلى أو إلى اليمين)، فيجري اختياره. الاستدعاء العَوْدِي الأخير -RECURSIVE-ACTIVITY SELECTOR$(s, f, 11, 11)$ يعيد Ø. المجموعة الناتجة عن النشاطات هي $\{a_1, a_4, a_8, a_{11}\}$.

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

يعمل الإجراء كما يلي: يشير المتحول k إلى أحدث إضافة إلى A، الموافقة للنشاط a_k في النسخة العَوْدِية. ولما كانت النشاطات مرتبة بحسب الترتيب المتزايد باطراد للحظات الانتهاء، فإن f_k هو دائمًا لحظة الانتهاء العُظْمَى لأي نشاط في A. أي إن:

$$f_k = \max \{f_i : a_i \in A\} \ . \qquad (3.16)$$

يختار السطران 2–3 النشاط a_1، ثم يجري استبداء A لتتضمن هذا النشاط فقط، واستبداء k ليؤشر إلى هذا النشاط. ثم تجد الحلقةُ **for** في الأسطر 4–7 النشاطَ الأبكر انتهاءً في S_k. تأخذ الحلقة كل نشاط a_m بالاعتبار، وتضيف a_m إلى A إذا كان متوافقًا مع جميع النشاطات المختارة سابقًا؛ مثل هذا النشاط هو أبكر نشاط ينتهي في S_k. ولمعرفة كون النشاط a_m متوافقًا مع جميع النشاطات الموجودة حاليًا في A، يكفي – اعتمادًا على المعادلة (3.16) – فحص لحظة البدء s_m (السطر 5)، والتأكد أنها ليست أبكر من f_k، زمن انتهاء آخر نشاط أُضيف إلى A. فإذا كان النشاط a_m متوافقًا، فإن السطرين 6–7 يضيفان هذا النشاط إلى A ويضعان القيمة m في k. إن المجموعة A التي أعادها الاستدعاء GREEDY-ACTIVITY-SELECTOR(s, f) هي تمامًا المجموعة التي أعادها الاستدعاء RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$.

يجدول الإجراء GREEDY-ACTIVITY-SELECTOR، شأنَ النسخة العَوْدِية، مجموعةً مكونة من n نشاطًا خلال زمن $\Theta(n)$، بافتراض أن النشاطات مفروزة منذ البداية تبعًا للحظات انتهائها.

تمارين

1-1.16

أعطِ خوارزميةَ برمجةٍ ديناميكية لحلِّ مسألة اختيار النشاطات، اعتمادًا على العلاقة العَوْدِية (2.16). اجعل خوارزميتك تحسب الحجوم $c[i,j]$ كما عُرِّفت سابقًا، وتُنتِج أيضًا المجموعةَ الجزئية ذات الحجم الأعظم من النشاطات المتوافقة فيما بينها. افترض أنه المدخلات فُرزت كما في المعادلة (1.16). قارن زمن تنفيذ حلِّك بزمن تنفيذ الخوارزمية GREEDY-ACTIVITY-SELECTOR.

2-1.16

افترض أننا عوضًا عن اختيارنا الدائم للنشاط الذي ينتهي أولاً، اخترنا آخر نشاط يبدأ ويكون متوافقًا مع جميع النشاطات المختارة سابقًا. بيّن كيف أن هذا النهج هو خوارزمية شرهة، وبرهن أنه يحقّق حلاً أمثل.

3-1.16

إن أي نهج شره لمسألة اختيار النشاطات لن يُنتِج مجموعةً ذات حجم أعظم من النشاطات المتوافقة فيما بينها. أعطِ مثالاً يبيِّن أن النهج المتمثّل في اختيار النشاط الأقصر من تلك النشاطات المتوافقة مع النشاطات المختارة سابقًا لن يعمل. أعِد الطلب نفسه للنهجَيْن: الأول الذي يختار دائمًا النشاط المتوافق الذي يتداخل مع أقل عدد مع النشاطات المتبقية، والثاني الذي يختار دائمًا النشاطَ المتبقي المتوافق ذا أبكر لحظة بدء.

4-1.16

لنفترض أن لدينا مجموعة نشاطات علينا جدولتها بين عدد كبير من قاعات المحاضرات، حيث يمكن أن يحصُل أيُّ نشاط في أية قاعة محاضرات. نودُّ جدولة جميع النشاطات باستخدام أقل عدد ممكن من قاعات المحاضرات. أعطِ خوارزمية شرهة فعالة لتحديد القاعة التي يستخدمها كل نشاط.

(تُعرَف هذه المسألة أيضًا ***بمسألة تلوين بيان–مجال interval-graph coloring problem***. يمكننا إنشاء بيان مجال عُقَدُهُ هي النشاطات المعطاة ووصلاتُهُ تربط النشاطات غير المتوافقة. إن أقل عدد من الألوان اللازمة لتلوين كل عقدة، بحيث لا نعطي لعقدتين متجاورتين اللون نفسه، يوافق إيجاد أقل عدد من قاعات المحاضرات لجدولة جميع النشاطات المعطاة.)

5-1.16

لندرس تعديلاً على مسألة اختيار النشاطات بحيث يكون لكل نشاط a_i قيمة v_i إضافة إلى لحظات البدء والانتهاء. ولم يَعُد الهدفُ جعْلَ عدد النشاطات المجدولة أعظميًّا، وإنما جعل القيمة الكلية للنشاطات المجدولة عُظْمَى عوضًا عن ذلك. أي علينا اختيار مجموعة A من النشاطات المتوافقة بحيث يكون $\sum_{a_k \in A} v_k$ أعظميًّا. أعطِ خوارزميةً تَحلُّ هذه المسألة بزمن كثير حدودي.

2.16 عناصر الاستراتيجية الشرهة

تتوصَّل الخوارزميةُ الشرهة إلى حلٍّ أمثلَ لمسألةٍ ما باتخاذ متتالية خيارات. لدى كل نقطة قرار في الخوارزمية، نختار الخيار الذي يبدو الأفضل عند تلك اللحظة. هذه الاستراتيجية الكسبية heuristic لا تُنتِج حلاً أمثلَ على الدوام، ولكن مثلما رأينا في مسألة اختيار النشاطات، يمكن أحيانًا أن تُنتِج حلاً أمثل. يناقش هذا المقطع بعض الخواص العامة للطرائق الشرهة.

لقد كانت الإجرائية التي اتبعناها في المقطع 1.16 لتطوير خوارزمية شرهة أكثر تعقيدًا بقليل من المعتاد.

لقد اتّبعنا الخطوات التالية:

1. تحديدُ البنية الجزئية المُثْلَى للمسألة.
2. تطويرُ حلٍّ عَوْدِي. (في مسألة اختيار النشاطات، قمنا بصياغة العلاقة العَوْدِية (2.16)، ولكننا لم نطور خوارزمية عَوْدِية تعتمد على تلك العلاقة.)
3. بيانُ أنه في حال اعتماد الخيار الشره، فإنه سيبقى مسألة جزئية وحيدة.
4. إثباتُ أن اعتماد الخيار الشره آمن دومًا (يمكن اعتماده دائمًا). (يمكن أن تحدث الخطوات 3 و 4 بأي ترتيب.)
5. تطويرُ خوارزمية عَوْدِية تنجّز الاستراتيجيةَ الشرهة.
6. تحويلُ الخوارزمية العَوْدِية إلى خوارزمية تكرارية.

باتباع هذه الخطوات، رأينا بتفصيل كبير دعائم البرمجة الديناميكية التي ترتكز عليها أية خوارزمية شرهة. على سبيل المثال، في مسألة اختيار النشاطات، عرّفنا أولاً المسائل الجزئية S_{ij}، حيث كان كل من i و j يتغيران. ثم وجدنا لاحقًا أننا إذا اعتمدنا الخيار الشره دائمًا، يمكن أن تقتصر مسائلنا الجزئية على الشكل S_k.

وبالمقابل، كان يمكننا تشكيل البنى الجزئية المُثْلَى على خلفية الخيار الشره، بحيث يترك الخيار مسألةً جزئية واحدة لحلِّها. في مسألة اختيار النشاطات، كان بإمكاننا الاستغناء عن الدليل الثاني وتعريف المسائل الجزئية من الشكل S_k. ثم كان بإمكاننا برهان أنه بتراكب الخيار الشره (أول نشاط a_m ينتهي في S_k)، مع حلٍّ أمثلَ لبقيةِ المجموعة S_m من النشاطات المتوافقة، نحصُل على حلٍّ أمثلَ لـ S_k. وبعمومية أكبر، نصمِّم الخوارزمياتِ الشرهةَ باتباع الخطوات التالية:

1. تحويلُ مسألة الأمثلة إلى مسألةٍ نتخذ فيها خيارًا، ويبقى علينا حلُّ مسألةٍ جزئيةٍ واحدة.
2. برهانُ وجودِ حلٍّ أمثلَ للمسألة الأصلية دائمًا، وهذا الحل يتخذ خيارًا شرهًا، بحيث يكون اتخاذ الخيار الشره آمنًا دائمًا.
3. إثباتُ البنى الجزئية المُثْلَى ببيان أنه باتخاذ الخيار الشره فإن ما يتبقى هو مسألة جزئية تتمتع بالخاصيّة التالية: إذا راكبنا الحل الأمثل للمسألة الجزئية مع الخيار الشره نتوصل إلى حل أمثل للمسألة الأصلية.

سنستخدم هذه الإجرائية المباشرة أكثر في مقاطع تالية من هذا الفصل. غير أنه يوجد في الغالب، خلف كل خوارزمية شرهة، حلٌّ أكثر تعقيدًا يعتمد البرمجة الديناميكية.

كيف يمكننا القول بأن خوارزميةً شرهةً ستَحُلُّ مسألةَ استمثالٍ (أمثَلَة) خاصة؟ لا توجد طريقة واحدةٌ تصلح لكل الحالات، غير أن خاصية الخيار الشره والبنى الجزئية المُثْلَى هما المكونان الأساسيان لها. فإذا استطعنا إثبات أن للمسألة هاتين الخاصيتين، فإننا على طريق تطوير خوارزمية شرهة للحل.

خاصيّة الخيار الشره

إن المكوّنَ الأساسي الأول هو ***خاصيّة الخيار الشره*** ***greedy-choice property***: يمكننا تجميع حلٍّ شاملٍ أمثلَ باتخاذ خيارٍ أمثلَ محليًّا (شره). وبعبارةٍ أخرى، حين نكون بصدد اعتماد أحد الخيارات، فإننا نعتمد الخيار الذي يبدو الأفضل في المسألة الحالية، دون الالتفات إلى نتائج المسائل الجزئية.

في هذه المرحلة، تختلف الخوارزميات الشرهة عن البرمجة الديناميكية. ففي البرمجة الديناميكية نعتمد خيارًا لدى كل خطوة، إلا أن الخيار يعتمد عادة على حلول المسائل الجزئية. نتيجة لذلك، فإننا نَحلُّ عادةً مسائلَ البرمجة الديناميكية بطريقة صعودية، متقدِّمين من مسائل جزئية صغرى إلى مسائل جزئية كبرى. (بالمقابل، يمكننا حلها نزوليًّا، ولكن باستذكار memoizing. ومع أن الرماز يعمل نزوليًّا، فما يزال علينا طبعًا حل المسائل الجزئية قبل اتخاذ الخيار.) في الخوارزمية الشرهة نأخذ أيَّ خيارٍ يبدو الأفضل في تلك اللحظة، ثم نَحلُّ المسائلَ الجزئية المتبقية. يمكن أن يَعتمد الخيار الذي نأخذه في الخوارزمية الشرهة على الخيارات الماضية، ولكنه لا يمكن أن يعتمد على أيِّ خياراتٍ مستقبلية أو على حلول المسائل الجزئية. وهكذا، وخلافًا للبرمجة الديناميكية، التي تحل المسائل الجزئية قبل اتخاذ الخيار الأول، فإن الخوارزمية الشرهة تتخذ خيارها الأول قبل حل أية مسألة جزئية. تتقدم خوارزمية البرمجة الديناميكية صعوديًّا، في حين تتقدم الاستراتيجية الشرهة عادةً نزوليًّا، بأخذ خيار شره واحد بعد الآخر، بحيث نقلص كل منتسخ instance للمسألة إلى مسألة أصغر منها.

علينا بالطبع أن نثبت أن الخيار الشره يعطي حلاً شاملاً أمثل، عند كل خطوة. وكما هو الحال في المبرهنة 1.16، فإن البرهان يدرس عادةً حلاً شاملاً أمثلَ لمسألةٍ جزئية. ثم يبيّن كيفية تعديل الحل للاستعاضة عن الخيار الشره بخيارات أخرى ينتج عنها مسائل جزئية مشابهة، ولكن أصغر من المسألة الأساسية.

يمكننا عادة اتخاذ الخيار الشره بفعالية أكثر من حالة اتخاذ مجموعةٍ أوسع من الخيارات. ففي مسألة اختيار النشاطات مثلاً، احتجنا لفحص كل نشاط مرة واحدة فقط، وذلك بفرض أننا فرزنا سلفًا النشاطات بحسب الترتيب المتزايد باطراد للحظات الانتهاء. بمعالجة بدائية للدخل أو باستخدام بنية معطيات مناسبة (وهي غالبًا رتل ذو أولوية)، يمكننا أخذ خيارات شرهة بسرعة، ونتوصّل بذلك إلى خوارزمية فعّالة.

بنيةٌ جزئيةٌ مُثْلَى

تُبدي مسألةٌ ما ***بنيةً جزئيةً مُثْلَى*** ***optimal substructure*** إذا تضمّن الحل الأمثل للمسألة حلولاً مُثْلَى لمسائل جزئية. هذه الخاصية هي إحدى المكونات الأساسية لتقدير قابلية تطبيق البرمجة الديناميكية أو الخوارزميات الشرهة. كمثال على البنى الجزئية المُثْلَى، نذكّر كيف برهنّا في المقطع 1.16، أنه إذا تضمّن حلٌّ أمثل لمسألة جزئية S_{ij} نشاطًا a_k، وجب عندها أن يتضمن أيضًا حلولاً مُثْلَى للمسائل الجزئية S_{ik} و S_{kj}. واعتمادًا على هذه البنية الجزئية المُثْلَى، ناقشنا أنه إذا علمنا أي نشاط علينا استعماله على أنه a_k، أمكننا بناء حلٍّ أمثلَ للمسألة S_{ij} باختيار النشاط a_k مع جميع النشاطات في حلول مُثْلَى للمسائل

الجزئية S_{ik}، و S_{kj}. وبالاعتماد على هذه الملاحظة التي تخص البنى الجزئية المُثْلَى، أمكننا استنباط العلاقة العَوْدِيّة (2.16) التي وَصَفت قيمةَ حلٍّ أمثل.

نستخدم عادة نهجًا أشدّ وضوحًا فيما يخص البنى الجزئية المُثْلَى حين نطبقها على الخوارزميات الشرهة. وكما ذكرنا آنفًا، أفرطنا في افتراض أننا وصلنا إلى مسألة جزئية (أي من الصيغة نفسها)، باتّخاذ خيار شره في المسألة الأصلية، على حين أن كل ما يلزمنا حقيقةً هو أن نناقش إذا كان ضمّ الحل الأمثل للمسألة الجزئية إلى الخيار الشره الذي المتَّخذ، يعطي حلاً أمثلَ للمسألة الأصلية. يَستخدم هذا المنهجُ ضمنيًّا الاستقراء على المسائل الجزئية لإثبات أن اتّخاذ الخيار الشره عند كل خطوة ينتج حلاً أمثل.

الخيار الشره مقابل البرمجة الديناميكية

لما كانت خاصيةُ البنى الجزئية المُثْلَى مستخدمةً في كلٍّ من الاستراتيجيات الشرهة والبرمجة الديناميكية، فقد ترغب في توليد حلٍّ بالبرمجة الديناميكية لمسألةٍ ما عندما يكون الحلُّ الشره كافيًا، أو بالعكس، تفكر خطأً في أن الحلَّ الشرهَ ناجعٌ في حين يتطلب الأمر في الحقيقة حلاً بالبرمجة الديناميكية. ولبيان الفوارق الدقيقة بين التقنيتين، سنتفحَّص نوعَيْن مختلفين لمسألةٍ أمثَلةٍ المعروفة.

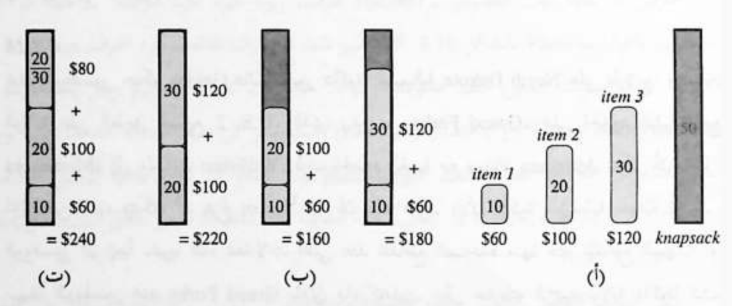
مسألة حقيبة الظهر 0-1 0-1 knapsack problem هي التالية: يسرق لصٌّ مخزنًا فيجد فيه n غرضًا؛ قيمة الغرض i هي v_i دولارًا ويزن w_i رطلاً، حيث v_i و w_i أعداد صحيحة. يريد اللصُّ أن يأخذ الحِمْلَ الأغلى ثمنًا قدر الإمكان، إلا أن بإمكانه حمل W رطلاً على الأكثر في حقيبة ظهره حيث W عدد صحيح ما. ما هي الأغراض التي ينبغي أن يأخذها؟ (سُمِّيت هذه المسألةُ مسألةَ حقيبة الظهر 0-1 لأن أيَّ غرض من الأغراض إما أن يأخذه اللصُّ وإما يدعه جانبًا؛ ولا يستطيع اللصُّ أخْذ جزءٍ من غرض، أو أخْذ غرضٍ أكثر من مرة واحدة.)

أما في ***مسألة حقيبة الظهر الكسرية fractional knapsack problem***، فالمشهد هو نفسه، غير أن اللصَّ يستطيع أخْذ كسور الأغراض، بدلاً من أن يكون لديه الخيار (0-1) لكل غرض. ويمكنك أن تتخيل الأغراض في مسألة حقيبة الظهر 0-1 على أنها سبائك ذهبية، أما الأغراض في مسألة حقيبة الظهر الكسرية فهي أكثر شبهًا بشذور الذهب.

تتمتع كلٌّ من مسألتَيْ حقيبة الظهر بخاصية البنى الجزئية المُثْلَى. لندرس، في المسألة 0-1، الحِمْلَ الأثمن الذي وزنه W رطلاً على الأكثر. إذا حذفنا الغرض j من الحِمْل فإن الحِمْلَ المتبقيَ هو الحِمْلُ الأثمن الذي يزن $W - w_j$ على الأكثر، الذي يستطيع اللص أخذه من $n - 1$ غرضًا أصليًّا باستبعاد j. وفي المسألة الكسرية المشابهة، نعتبر أننا إذا حذفنا وزنًا w من أحد العناصر j من الحِمْل الأمثل، وَجَبَ أن يكون الحِمْلُ المتبقي هو الحِمْلَ الأثْمنَ ذا الوزن $W - w$ على الأكثر، الذي يستطيع اللص أخذه من الأغراض الأصلية $n - 1$ إضافة إلى $w_j - w$ رطلاً من الغرض j.

ومع أن المسألتَيْن متشابهتان، فيمكننا حلُّ مسألة حقيبة الظهر الكسرية باستخدام استراتيجية شرهة، ولكننا لا نستطيع حلّ المسألة 1-0 بالاستراتيجية نفسها. ولحلّ المسألة الكسرية، نحسب أولاً قيمةَ الرطل الواحد v_i/w_i لكل غرض. وباتباع استراتيجية شرهة، يبدأ اللص بأخذ ما يمكن من الغرض ذي القيمة العليا للرطل الواحد. فإذا نَفِدَ ذلك الغرض، وكان بإمكان اللص أن يحمل أكثر، فإنه يأخذ ما يُمكِنُهُ أخْذُهُ من الغرض ذي القيمة العليا التالية للرطل الواحد، وهكذا، إلى أن لا يعود بإمكانه حَمْل أيّ شيءٍ زائد. وهكذا، وبفرز الأغراض بحسب قيمة الرطل الواحد لكل منها، تُنفَّذ الخوارزمية الشرهة بزمن $O(n \lg n)$. وسيُطلب إليك في التمرين 2.16-1 البرهان على أن مسألة حقيبة الظهر الكسرية تتمتع بخاصيّة الخيار الشره.

ولمعرفة أن الاستراتيجية الشرهة لا تعمل في مسألة حقيبة الظهر 1-0، نأخذ منتسَخَ instance المسألة المبين في الشكل 2.16(أ). ففي هذا المثال ثلاثة عناصر، وبإمكان حقيبة الظهر حَمْل 50 رطلاً. العنصر 1 يزن 10 أرطال وقيمته 60 دولارًا، والعنصر 2 يزن 20 رطلاً وقيمته 100 دولار، والعنصر 3 يزن 30 رطلاً وقيمته 120 دولارًا. وهكذا، فإن قيمة كل رطل من العنصر 1 تساوي 6 دولارت، وهي أكبر من قيمة الرطل لأي من العنصر 2 (5 دولارات لكل رطل) أو العنصر 3 (4 دولارات لكل رطل). لذلك، فإن الاستراتيجية الشرهة ستأخذ العنصر 1 أولاً. ولكن، كما يظهر من تحليل الحالة في الشكل 2.16(ب)، فإن الحلَّ الأمثلَ يأخذ العنصرَيْن 2 و 3 تاركًا العنصر 1. أما الحلاَّن الممكنان الآخران اللذان يأخذان العنصر 1 بالاعتبار فهما غير أمثليَّيْن.



الشكل 2.16 مثالٌ يبيِّن أن الاستراتيجية الشرهة لا تناسب حالة مسألة حقيبة الظهر 1-0. (أ) على اللص أن يختار مجموعةً جزئية من الأغراض الثلاثة المبينة بحيث لا يزيد وزنُها الكلي عن 50 رطلاً. (ب) تتضمن المجموعة الجزئية المُثْلَى الغرضَيْن 2 و 3. وأيُّ حلٍّ يتضمَّن الغرضَ 1 هو حلٌّ غيرُ أمثل (أمثل جزئيًّا)، بالرغم من أن للغرض 1 أعلى قيمة للرطل الواحد. (ت) في حالة مسألة حقيبة الظهر الكسرية، فإن أخْذ الأغراض بحسب ترتيبِ أعلى قيمة لكل رطل يعطي حلاًّ أمثل.

ولكن، في المسألة الكسرية المشابهة، تؤدي الاستراتيجية الشرهة التي تأخذ العنصر 1 أولاً إلى حلٍّ أمثل، كما هو مبيّن في الشكل 2.16(ت). فيما لا يمكن أخذ العنصر 1 في المسألة 1-0 لأن ذلك يمنع اللص من ملء حقيبة ظهره بكل سعتها، والفراغ في الحقيبة يخفِّض القيمةَ الفعلية للرطل الواحد من حمله. حين نأخذ بالاعتبار عنصرًا لتضمينه في الحقيبة الظهرية في المسألة 1-0، علينا أن نقارن حلَّ المسألة الجزئية الذي يضمّن هذا العنصر بحلِّ المسألة الجزئية الذي يَستبعد هذا العنصر قبل اعتماد خيارنا. ينتج عن صياغة المسألة بهذه الطريقة، مسائل جزئية كثيرة متراكبة، وهذه سمةٌ مميِّزةٌ للبرمجة الديناميكية، وبالفعل، مثلما يُطلب إليك بيانه في التمرين 2.16-2، يمكن استخدام البرمجة الديناميكية لحل المسألة 1-0.

تمارين

1-2.16

برهن أن مسألة حقيبة الظهر الكسرية تتمتع بخاصية الخيار الشره.

2-2.16

أعطِ حلاً بالبرمجة الديناميكية لمسألة حقيبة الظهر 1-0 تُنفَّذ بزمن $O(n\,W)$، حيث n عدد العناصر، و W الوزن الأعظم للأغراض التي يستطيع اللص وضعها في حقيبة ظهره.

3-2.16

افترض في مسألة حقيبة الظهر 1-0 أن ترتيب الأغراض حين فرزها تصاعديًّا تبعًا للوزن، هو نفسه ترتيبها بحسب قيمها المتناقصة. أعطِ خوارزميةً فعالةً لإيجاد الحل الأمثل لهذا الشكل المعدَّل من مسألة حقيبة الظهر، وناقش صحة خوارزميتك.

4-2.16

يحلم البروفسور جيكو Gekko دائمًا بعبور داكوتا الشمالية North Dakota على المزلاج. وهو يخطِّط لعبور الولاية على الطريق السريع U.S. 2، الذي يبدأ من Grand Forks، على الحدود الشرقية مع مينيسوتا Minnesota، إلى ولستون Williston، قرب الحدود الغربية مع مونتانا Montana. يمكن أن يحمل البروفسور لترَيْن من الماء، ويمكنه أن يتزلج m ميلاً قبل أن ينفد ماؤه. (لأن داكوتا الشمالية سهليّة نسبيًّا، ليس على البروفسور أن يَعبأ بشربه الماء بمعدلات أعلى عند المقاطع الصاعدة منها عند المقاطع السهلية أو الهابطة.) سيبدأ البروفسور عند Grand Forks بلترَيْ ماء كاملين. تبيِّن خارطته الرسمية لولاية داكوتا الشمالية جميع الأماكن على طول الطريق U.S. 2 التي يمكنه عندها إعادة ملء الماء، والمسافات بين هذه المواضع.

هدف البروفسور هو تصغير عدد التوقفات المائية على طول الطريق عبر الولاية. أعطِ طريقة فعّالة يستطيع بواسطتها تحديد التوقفات التي يجب أن يقوم بها لملء الماء. بيّن أن استراتيجيتك تحقِّق حلاً أمثل، وأعطِ زمن التنفيذ.

5-2.16

صِفْ خوارزميةً فعالة، تأخذ مجموعة من النقاط $\{x_1, x_2, \ldots, x_n\}$ على المستقيم الحقيقي، وتحدِّد أصغر مجموعة من المجالات المغلقة التي طولها واحد تحتوي كل النقاط المعطاة. ناقش صحة خوارزميتك.

★ *6-2.16*

بيّن كيف تحل مسألة حقيبة الظهر الكسرية بزمن $O(n)$.

7-2.16

افترض أنَّ لديك مجموعتَيْن A و B، تتضمن كلٌّ منها n عددًا صحيحًا موجبًا. يمكنك إعادة ترتيب كل مجموعة بالطريقة التي تريد. بعد إعادة الترتيب، ليكن a_i العنصرَ ذا الترتيب i من المجموعة A، و b_i العنصرَ ذا العنصر i من المجموعة B. بعد ذلك ستحصل على مكافأة بقيمة $\prod_{i=1}^{n} a_i^{b_i}$. أعطِ خوارزميةً تجعل المكافأة عُظْمَى، أثبت أن خوارزميتك تجعل المكافأة عُظْمَى، وصرّح عن زمن التنفيذ.

3.16 أرمزة هوفمان

تضغط أرمزة هوفمان المعطيات بفعالية عالية جدًّا؛ ويمكن عادةً تحقيق وفر بنسبة 20% إلى 90%، وذلك تبعًا لخواص المعطيات التي هي في قيد الضغط. نفترض أن المعطيات هي متتالية من المحارف. تَستخدم خوارزميةُ هوفمان الشرهة جدولاً بمرّات ورود المحارف (أي تواتراتها) لبناء طريقة مُثْلَى لتمثيل كل محرف كسلسلة محارف اثنانية.

افترض أن لدينا ملفَّ معطياتٍ بـ 100,000 محرف، ونود خزنه خزنًا متراصًا. نلاحظ أن المحارف في الملف تَرِد بالتواترات المعطاة بالشكل 3.16. أي تَظهر فقط 6 محارف مختلفة، ويَرِد المحرف a ‏45,000 مرة.

ثمة عدة خيارات لتمثيل ملف معلومات كهذا. هنا، نهتم بمسألة تصميم ***رماز بالمحارف الاثنانية*** ***binary character code*** (أو باختصار ***رماز code***) حيث نرمّز كل محرف بمتتالية محارف اثنانية وحيدة. إذا استخدمنا ***رمازًا محدَّد الطول fixed-length code*** فإننا نحتاج إلى ثلاثة بتات لتمثيل المحارف الستة: a = 000, b = 001, ..., f = 101. تتطلب هذه الطريقة 300,000 بتًّا لترميز كامل الملف. هل يمكننا القيام بما هو أفضل من هذا؟

يمكن ***للرماز المتغير الطول variable-length code*** أن يكون أداؤه أفضل بكثير من الرماز المحدد الطول، بإعطاء المحارف العالية التواتر كلمات رماز قصيرة والمحارف النادرة كلمات رماز طويلة. يبيّن الشكل 3.16 هذا الرماز؛ هنا، يمثل المحرف ذو البت الوحيد 0 المحرف a، ويمثل المحرف ذي البتات الأربعة 1100 المحرف f. يتطلب هذا الرماز:

$$[(45)(1) + (13)(3) + (12)(3) + (16)(3) + (9)(4) + (5)(4)](1{,}000) = 224{,}000 \text{ بت}$$

المحرف	a	b	c	d	e	f
تواتره (بالآلاف)	45	13	12	16	9	5
كلمة الرماز بطول ثابت	000	001	010	011	100	101
كلمة الرماز بطول متغير	0	101	100	111	1101	1100

الشكل 3.16 مسألة ترميز محارف. يتضمن ملف معطيات 100,000 محرفًا، من مجموعة المحارف a-f فقط بالتواترات المبينة. إذا أسندنا رمازًا من ثلاثة بتات لكل محرف، فيمكن ترميز الملف بـ 300,000 بت. أما باستخدام الرماز المتغير الطول المبين، فيمكن ترميز الملف بـ 224,000 بت.

لتمثيل الملف، أي بوفْر بنسبة 25% تقريبًا. وهو في الواقع رماز أمثل للمحارف لهذا الملف، مثلما سنرى لاحقًا.

الأرمزة السبقية

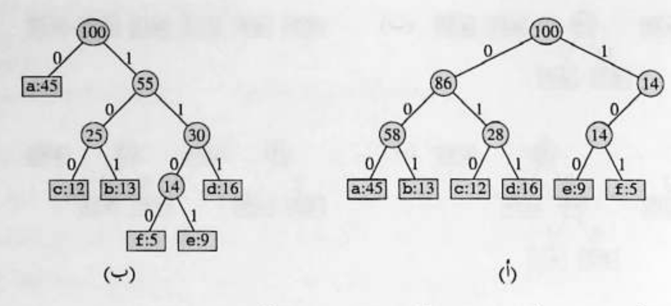
سندرس هنا، فقط الأرمزة التي لا تكون فيها أية كلمة رماز سابقةً prefix لكلمة رماز أخرى. تسمى هذه الأرمزة *أرمزة سبقية*[3] ***Prefix codes***. يمكن أن نبيِّن، علمًا بأننا لن نبرهن ذلك هنا، أن الرماز السبقي يمكنه دائمًا إنجاز ضغط المعطيات الأمثل من بين أي رماز للمحارف، وبذلك فإننا لا ننقص من عمومية المسألة إذا قصرنا اهتمامنا على الأرمزة السبقية.

إن عملية الترميز بسيطة لأي رماز محارف اثنانية؛ إذ يكفي ضم كلمات الرماز التي تمثل كل محرف في الملف. على سبيل المثال، نرمِّز ملف المحارف الثلاثة abc بالرماز المتغير الطول السبقي الموضح بالشكل 3.16 كما يلي: $0 \cdot 101 \cdot 100 = 0101100$، حيث يشير "$\cdot$" إلى الضم.

الأرمزة السبقية مرغوبة لأنه يمكن فك ترميزها ببساطة. ولما كانت أية كلمة رماز لا تكوِّن سابقةً لأية كلمة رماز أخرى، فإن كلمة الرماز التي يبدأ بها ملف مرمَّز غير مُلْبِسَة. يمكننا ببساطة تحديد كلمة الرماز الأولى، وإعادتها إلى المحرف الأصلي، وتكرار عملية فك ترميز بقية الملف المرمز. في مثالنا، تحلَّل سلسلة المحارف 001011101 تحليلاً وحيدًا إلى $0 \cdot 0 \cdot 101 \cdot 1101$، ويُفكّ ترميزها إلى aabe.

تحتاج عملية فك الترميز إلى تمثيل مناسب للأرمزة السبقية، بحيث يمكننا التقاط كلمة الرماز الأولى بسهولة. توفر الشجرة الثنائية التي أوراقها هي المحارف المعطاة مثل هذا التمثيل. نفسر الكلمة الاثنانية لمحرف على أنها المسار البسيط من الجذر إلى ذلك المحرف، بحيث يعني 0 "اذهب إلى الابن الأيسر" و 1 "اذهب إلى الابن الأيمن". يبيّن الشكل 4.16 أشجار الرمازين في مثالنا. لاحظ أن هذه الأشجار ليست أشجار بحث ثنائية، لأنها لا تتطلب ظهور الأوراق بترتيب مفروز، ولأن العقد الداخلية لا تتضمن مفاتيح محرفية.

[3] ربما تكون التسمية "أرمزة من دون سوابق" أفضل، إلا أن "الأرمزة السبقية" معيارية في المراجع.



الشكل 4.16 الأشجار الموافقة لأساليب الترميز في الشكل 3.16. كل ورقة لها لصيقة بالمحرف مع تواتر وروده، وكل عقدة داخلية لها لصيقة بمجموع تواتر ورود الأوراق في شجرتها الفرعية. (أ) الشجرة الموافقة للرماز المحدد الطول a = 000, ..., f = 101. (**ب**) الشجرة الموافقة للرماز السبقي الأمثل a = 0, b = 101, ..., f = 1100.

يجري دائمًا تمثيل الرماز الأمثل لملف بشجرة ثنائية *ملأى full*، بحيث يكون لكل عقدة إن لم تكن ورقة ابنان (انظر التمرين 3.16-2). إن الرماز المحدد الطول في مثالنا ليس أمثليًا، لأن شجرته المبينة في الشكل 4.16(أ) ليست شجرة ثنائية ملأى: ثمة كلمات رماز تبدأ بـ ...10 ولكن لا تبدأ أي كلمة رماز بـ ...11. ولأننا سنقصر اهتمامنا الآن على الأشجار الثنائية الملأى، يمكننا القول إنه إذا كانت C الأبجدية التي تنتمي إليها المحارف، وكانت تواترات جميع المحارف موجبة، كان لأشجار الأرمزة السبقية المُثْلَى $|C|$ ورقة بالضبط، ورقة لكل حرف في الأبجدية، وكان لها تمامًا $|C| - 1$ عقدة داخلية (انظر التمرين 5-3.ب).

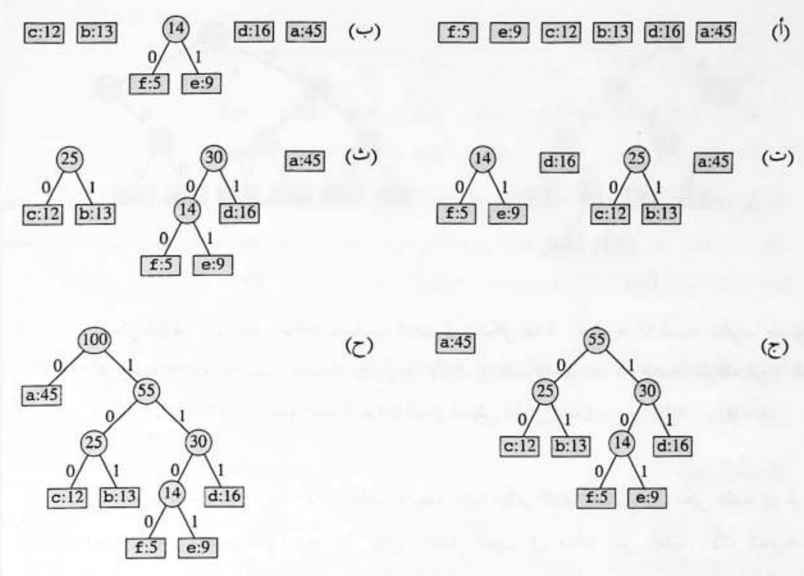
إذا كان لدينا شجرة T توافق رمازًا سبقيًا، يمكن بسهولة عدّ البتات اللازمة لترميز الملف. لنرمز بالواصفة $c.freq$ إلى تواتر المحرف c من الأبجدية C في الملف، وليكن $d_T(c)$ عمق ورقة المحرف c في الشجرة. لاحظ أن $d_T(c)$ هو أيضًا طول كلمة ترميز المحرف c. وبذلك، يكون عدد البتات التي يتطلبها ترميز الملف هو:

$$B(T) = \sum_{c \in C} c.freq.d_T(c) \ , \qquad (4.16)$$

ونعرّف هذا العدد بأنه ***كلفة cost*** الشجرة T.

بناء رماز هوفمان

اخترع هوفمان خوارزمية شرهة تبني رمازًا سبقيًا أمثلَ يسمى ***رماز هوفمان Huffman code***. وتماشيًا مع ملاحظاتنا في المقطع 2.16، تعتمد صحة هذا الرماز على خاصية الخيار الشره والبنى الجزئية المُثْلَى. وعوضًا عن تبيان تحقق هذه الخواص ثم تطوير شبه الرماز، فإننا سنعرض شبه الرماز أولاً. إذ سيساعد هذا في توضيح كيفية قيام الخوارزمية بالخيارات الشرهة.



الشكل 5.16 خطوات خوارزمية هوفمان للتواترات المعطاة في الشكل 3.16. يبيِّن كلُّ جزءٍ محتويات الرتل مفروزةً تصاعديًّا تبعًا للتواتر. في كلِّ خطوة، تُدمَج الورقتان الأقل تواترًا. تَظهر الأوراقُ على شكل مستطيلات تحتوي المحرفَ وتواترَه، والعُقَدُ الداخليةُ على شكل دوائر تتضمن مجموع تواترات أبنائها. توضع لصاقةٌ على الوصلة بين العقد الداخلية وأبنائها، قيمتُها 0 إذا كانت الوصلة إلى الابن الأيسر، و 1 إذا كانت الوصلة إلى الابن الأيمن. إن كلمة رماز محرف هي متتالية اللصاقات على الوصلات التي تربط الجذر بورقة ذلك الحرف. (أ) المجموعة البدائية من $n = 6$ عقد، عقدة لكل حرف. (ب)–(ج) مراحل وسيطية. (ح) الشجرة النهائية.

نفترض في شبه الرماز التالي أن C مجموعةٌ من n محرفًا، وأن كلَّ محرفٍ $c \in C$ هو غرضٌ له واصفةٌ تعطي تواترَه $c.freq$. تَبني الخوارزميةُ الشجرةَ T الموافقةَ للرماز الأمثل بطريقة صعودية. فهي تبدأ بمجموعةٍ من $|C|$ ورقةً، ثم تنجز متتالية من $|C| - 1$ عمليةَ "دمجٍ" لإنشاء الشجرة النهائية. تَستخدم الخوارزميةُ رتلاً ذا أولوية الأصغر Q، مفتاحُهُ الواصفةُ $freq$، لتعيين الغرضَيْن ذوي التواتر الأقل لدمجهما. إن ناتجَ الدمج هو غرضٌ جديدٌ تواترُهُ هو مجموع تواترَي الغرضَيْن اللذين دُمِجَا.

HUFFMAN(C)

1 $n = |C|$
2 $Q = C$
3 **for** $i = 1$ **to** $n - 1$
4 allocate a new node z
5 $z.left = x =$ EXTRACT-MIN(Q)

```
6        z.right = y = EXTRACT-MIN(Q)
7        z.freq = x.freq + y.freq
8        INSERT(Q, z)
9        return EXTRACT-MIN(Q)     // return the root of the tree
```

تعمل خوارزميةُ هوفمان في مثالنا كما هو مبيَّن في الشكل 5.16. ولما كانت الأبجديةُ تتضمن 6 محارف، فإن حجم الرتل البدائي $n = 6$، ويلزم 5 خطواتِ دمجٍ لبناءِ الشجرة. تمثِّل الشجرةُ النهائية الرمازَ السبقي الأمثل. إن كلمة الرماز لحرف هي متتالية لصيقات الوصلات على المسار البسيط من الجذر إلى الحرف.

يستبدئ السطر 2 رتل ذو أولوية الأصغر Q بمله محارف C. تَستخرج حلقة **for**، في الأسطر 3-8، تكراريًّا، العقدتَيْن x و y الأدنى تواترًا في الرتل، وتستعيض عنهما في الرتل بعقدة جديدة z تمثِّل ناتج دمجهما. يُحسَب تواتر z في السطر 7 على أنه مجموع تواتري x و y. للعقدة z ابنان، x هو الابن الأيسر و y هو الابن الأيمن. (هذا الترتيب اعتباطي؛ فالتبديل بين الابن الأيسر والأيمن لأية عقدة يُنتِج رمازًا مختلفًا ولكن له الكلفة نفسها.) بعد $n - 1$ عملية دمج، يعيد السطر 9 العقدة الوحيدة المتبقية في الرتل، التي هي جذر شجرة الرماز.

ومع أن الخوارزمية ستعطي نفس النتيجة لو أننا أزلنا المتحولين x و y (بإسناد القيم مباشرة إلى $z.left$ و $z.right$ في السطرين 5 و 6، وتغيير السطر 7 إلى $z.freq = z.left.freq + z.right.freq$)، فإننا سنستخدم الاسمين x و y لبرهان صحة الخوارزمية. لذلك، رأينا من المناسب الإبقاء عليهما.

لتحليل زمن تنفيذ خوارزمية هوفمان نفترض أن الرتل Q منجَّز ككومة اثنانية وفق الأصغر binary min-heap (انظر الفصل 6). في حال تكونت المجموعة C من n محرفًا، يمكننا إنجاز استبداء الرتل Q في السطر 2 بزمن $O(n)$ باستخدام الإجراء BUILD-MIN-HEAP الذي ناقشناه في المقطع 3.6. تُنفَّذ حلقة **for** في الأسطر 3-8 $n - 1$ مرةً تمامًا. ولما كانت كلُّ عملية كومة تتطلب زمنًا $O(\lg n)$، فإن الحلقة تساهم بزمن $O(n \lg n)$ في زمن التنفيذ. وبذلك يكون زمن التنفيذ الكلي لـ HUFFMAN على مجموعةٍ من n محرفًا هو $O(n \lg n)$. يمكننا خفض زمن التنفيذ إلى $O(n \lg \lg n)$ بالاستعاضة عن الكومة الاثنانية وفق الأصغر بشجرة van Emde Boas (انظر الفصل 20).

صحة خوارزمية هوفمان

لإثبات أن خوارزمية HUFFMAN الشرهة صحيحة، سنبيِّن أن مسألة تعيين رماز سبقي أمثل تحقق خاصيتَي الخيار الشره والبنية الجزئية المُثلَى. تبيِّن التوطئة التالية أن خاصية الخيار الشره محقَّقة.

توطئة 2.16

إذا كانت C أبجدية بحيث يكون لكلِّ محرفٍ $c \in C$ تواترٌ $c.freq$، وإذا كان x و y محرفَيْن في C لهما أصغر

التواترات، فيوجد رمازٌ سبقيٌّ أمثل optimal prefix code لـ C بحيث يكون لكلمتي الرماز الخاصتَيْن بـ x و y الطولُ نفسُه، وتختلفان في البت الأخير فقط.

البرهان تكمن فكرة البرهان في أخذ الشجرة T التي تمثل رمازًا سبقيًّا أمثلَ اعتباطيًّا، وتعديلها لبناء شجرةٍ تمثِّل رمازًا سبقيًّا أمثلَ آخر بحيث يَظهر المحرفان x و y كورقتَيْن أختين لهما أعظم عمق في الشجرة الجديدة. إذا استطعنا بناء مثل هذه الشجرة، فسيكون لكلمتي الرماز الخاصتين بـ x و y الطولُ نفسُه، وستختلفان في البت الأخير.

ليكن المحرفان a و b ورقتَيْن أختَيْن لهما العمق الأعظم في T. ولكي لا نفقد عمومية الحل، نفترض أن $a.freq \leq b.freq$ و $x.freq \leq y.freq$. ولما كان $x.freq$ و $y.freq$ أصغر تواترين مرتبين للأوراق، وكان $a.freq$ و $b.freq$ تواترَيْن اعتباطيين مرتبين، فإن $x.freq \leq a.freq$ و $y.freq \leq b.freq$.

في بقية البرهان، من الممكن أن يكون لدينا $x.freq = a.freq$ أو $y.freq = b.freq$. ومع ذلك، إذا كان $x.freq = b.freq$ فسيكون لدينا أيضًا $a.freq = b.freq = x.freq = y.freq$ (انظر التمرين 16.3-1)، وستكون التوطئة مبَرهَنة بداهةً. لذلك سنفترض أن $x.freq \neq b.freq$، وهذا يعني أن $x \neq b$.

كما يبيّن الشكل 6.16، نبادل موضعَي a و x في T لإنتاج شجرة T'، ثم نبادل موضعَي b و y في T' لإنتاج شجرة T'' تكون فيها الورقتان x و y أختَيْن بعمقٍ أعظم. (لاحظ أنه إذا كان $x = b$ ولكن $y \neq a$، عندها لن تكون x و y أختَين في الشجرة T'' بعمقٍ أعظم. ولأننا نفترض أن $x \neq b$، فإن هذه الحالة لن تحصُل.) من المعادلة (4.16) يكون فرق التكلفة بين T و T' هو:

$$\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} c.freq.d_T(c) - \sum_{c \in C} c.freq.d_{T'}(c) \\
&= x.freq.d_T(x) + a.freq.d_T(a) - x.freq.d_{T'}(x) - a.freq.d_{T'}(a) \\
&= x.freq.d_T(x) + a.freq.d_T(a) - x.freq.d_T(a) - a.freq.d_T(x) \\
&= (a.freq - x.freq)(d_T(a) - d_T(x)) \\
&\geq 0 \,,
\end{aligned}$$

لأن كلاًّ من $a.freq - x.freq$ و $d_T(a) - d_T(x)$ غير سالب. وبتخصيصٍ أدق، $a.freq - x.freq$ غير سالب لأن x هي ورقة ذات أصغر تواتر، و $d_T(a) - d_T(x)$ غير سالب لأن a ورقة ذات عمقٍ أعظم في T. وبالمثل، فإن المبادلة بين y و b لا يزيد التكلفة، وبذلك يكون $B(T') - B(T'')$ غير سالب. ولذلك، فإن $B(T'') \leq B(T)$، ولما كانت T مُثلَى، فإن $B(T) \leq B(T'')$، وهذا يقتضي أن $B(T'') = B(T)$. إذن، T'' شجرة مُثلَى تظهر فيها x و y كورقتَيْن أختين لهما عمق أعظم، ومنه نستنتج صحة التوطئة. ■

الشكل 6.16 توضيح الخطوة المفتاحية في برهان التوطئة 2.16. في الشجرة المُثْلَى T، الورقتان a و b هما أختان تتمتعان بعمقٍ أعظم. الورقتان x و y هما الحرفان اللذان لهما التواتر الأقل؛ تظهران بموضعين اعتباطيين في الشجرة T. بافتراض أن $x \neq b$، فإن إبدال الورقتين a و x فيما بينهما يُنتِج الشجرة T'، ثم يُنتِج إبدال الورقتين b و y الشجرةَ T''. ولما كانت أية عملية إبدال لا تزيد في الكلفة، فإن الشجرةَ الناتجة T'' هي أيضًا شجرةٌ مُثْلَى.

تقتضي التوطئة 2.16 أنه يمكن، دون المساس بعمومية المسألة، بَدْءُ إجرائية بناء شجرة مُثْلَى باستخدام الخيار الشره الذي يدمج الحرفين ذوي التواتر الأقل. لماذا هذا الخيار هو خيار شره؟ يمكننا رؤية تكلفة عملية دمج وحيد على أنها مجموع تواترَي العنصرين المدموجين. يبيّن التمرين 3.16-4 أن التكلفة الكلية للشجرة التي جرى بناؤها يساوي مجموع تكلفة عمليات الدمج. من بين جميع عمليات الدمج الممكنة، عند كل خطوة، تستهدف خوارزمية HUFFMAN عملية الدمج ذات التكلفة الأقل.

تبيّن التوطئة التالية أن لمسألة بناء أرمزة سبقية مُثْلَى خاصية البنية الجزئية المُثْلَى.

توطئة 3.16

لتكن C أبجدية بتواتر $c.freq$ لكلِّ محرف $c \in C$. وليكن x و y محرفين من C بأقل التواترات. ولتكن الأبجدية C' التي حُذِف منها x و y، وأُضيف إليها محرف جديد z، أي: $C' = C - \{x, y\} \cup \{z\}$. نعرّف $freq$ للأبجدية C' كما هو معرّف لـ C باستثناء أن $z.freq = x.freq + y.freq$. لتكن T' أية شجرة تمثِّل رمازًا سبقيًّا أمثلَ للأبجدية C'. عندها تمثِّل الشجرة T، الناتجة من T' بالاستعاضة عن عقدة الورقة z بعقدة داخلية أبناؤها x و y، رمازًا سبقيًّا أمثلَ للأبجدية C.

البرهان نبيِّن أولاً أنه يمكن التعبير عن $B(T)$ تكلفة الشجرة T بدلالة $B(T')$ تكلفة الشجرة T' بالأخذ بالحسبان تكاليف مكونات المعادلة (4.16). لكلِّ محرف $c \in C - \{x, y\}$، لدينا $d_T(c) = d_{T'}(c)$، ولذلك، فإن $c.freq.d_T(c) = c.freq.d_{T'}(c)$. ولما كان $d_T(x) = d_T(y) = d_{T'}(z) + 1$، فيكون لدينا:

$$
\begin{aligned}
x.freq.d_T(x) + y.freqd_T(y) &= (x.freq + y.freq)(d_{T'}(z) + 1) \\
&= z.freq.d_{T'}(z) + (x.freq + y.freq) \ ,
\end{aligned}
$$

ومنه نستنتج أن:

$$B(T) = B(T') + x.freq + y.freq$$

أو العلاقة المكافئة:

$$B(T') = B(T) - x.freq - y.freq \ .$$

سنبرهن التوطئة الآن بطريقة نقض الفرض. لنفترض أن T لا تُمثِّل رمازًا سبقيًّا أمثلَ للأبجدية C. إذن توجد شجرةٌ مُثْلَى T'' بحيث $B(T'') < B(T)$. ومن دون إنقاص عمومية المسألة (باستخدام التوطئة 2.16)، نعتبر x و y أختين في الشجرة T''. لتكن T''' الشجرة T'' بالاستعاضة عن الأب المشترك لـ x و y بالورقة z بتواتر $z.freq = x.freq + y.freq$. وعندها يكون:

$$\begin{aligned} B(T''') &= B(T'') - x.freq - y.freq \\ &< B(T) - x.freq - y.freq \\ &= B(T') \ , \end{aligned}$$

وهذا يناقض الفرض أن T' تمثل رمازًا سبقيًّا أمثل لـ C'. وبذلك، يجب أن تمثِّل T رمازًا سبقيًّا أمثل للأبجدية C. ■

مبرهنة 4.16

تُنتِج الإجرائيةُ HUFFMAN رمازًا سبقيًّا أمثل.

البرهان يمكن إثبات هذه المبرهنة مباشرةً من التوطئتين 2.16 و 3.16. ■

تمارين

1-3.16

ذكرنا في برهان التوطئة 2.16 أنه إذا كان $x.freq = b.freq$ وَجَبَ أن يكون $a.freq = b.freq = x.freq = y.freq$. وضِّح السبب.

2-3.16

برهن أنه لا يمكن أن توافق شجرةٌ ثنائية غير ملأى رمازًا سبقيًّا أمثل.

3-3.16

ما هو رماز هوفمان الأمثل للمجموعة التالية من الترددات، التي تعتمد على أعداد فيبوناتشي Fibonacci الثمانية الأولى؟

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

هل يمكنك تعميم إجابتك لإيجاد الرماز الأمثل حين تكون التواترات هي أعداد فيبوناتشي الـ n الأولى؟

4-3.16

برهن أنه يمكن أيضًا حساب التكلفة الكلية لشجرة رماز، على أنها مجموع تواترَي ابنَي كل عقدة داخلية.

5-3.16

برهن أنه إذا رتبنا محارف أبجدية بحيث تكون تواتراتها متناقصة باطراد، فثمة رماز أمثل تكون كلمات رمازه متزايدة الطول باطراد.

6-3.16

لنفترض أن لدينا رمازًا سبقيًّا أمثلَ على مجموعة $C = \{0, 1, \ldots, n-1\}$ من المحارف، وأننا نود إرسال هذا الرماز باستخدام أقل عدد ممكن من البتات. بيّن كيف يمكن تمثيل أي رماز سبقي أمثل على C باستخدام $2n - 1 + n\lceil \lg n \rceil$ بتًّا فقط. (*تلميح*: استخدم $2n - 1$ بتًّا لتحديد بنية الشجرة، فيما يجري استكشافها بالمسير عبرها.)

7-3.16

عمِّم خوارزمية هوفمان لكلمات رماز ثلاثية (أي كلمات رماز تستخدم الرموز 0 و 1 و 2)، برهن أنها تعطي أرمزة ثلاثية مُثْلَى.

8-3.16

افترض أن لدينا ملفَّ معطياتٍ يتضمن متتاليةً من محارف ذات ثمانية بتات بحيث تكون جميع المحارف الـ 256 تقريبًا بنفس الشيوع: أي التواتر الأعظم للمحارف أقل من ضعف التواتر الأدنى لها. برهن أن ترميز هوفمان في هذه الحالة ليس أكثر فعالية من الرماز العادي المحدد الطول بـ 8 بتات.

9-3.16

بيّن أنه لا يوجد أسلوب ضغط يُتوقَّع أن يضغط ملف محارف ذات ثمانية بتات مختارة عشوائيًّا ولا حتى ليقلّص منه بتًّا واحدًا. (*تلميح*: قارن عدد الملفات بعدد الملفات المرمّزة الممكنة.)

★ 4.16 الكيانات المصفوفية والطرائق الشرهة

نَعرض في هذا المقطع نظريةً جميلة عن الخوارزميات الشرهة. تصف هذه النظرية حالات كثيرة تعطي فيها الطريقةُ الشرهةُ حلولاً مُثْلَى. وهي تتطلب بنى تراكبية (توافقية) تسمى كيانات مصفوفية "matroids". ومع أن هذه النظرية لا تشمل جميع الحالات التي تُطبَّق فيها الطرائق الشرهة (فهي لا تشمل مثلاً مسألة اختيار النشاط في المقطع 1.16 أو مسألة ترميز هوفمان في المقطع 3.16)، إلا أنها تشمل الكثير من الحالات الهامة عمليًّا. يضاف إلى ذلك، أن هذه النظرية توسَّعت لتشمل تطبيقات عديدة كثيرة؛ انظر الملاحظات في آخر هذا الفصل لمعرفة المراجع.

الكيانات المصفوفية

الكيان المصفوفي Matroid هو زوج مرتب $M = (S, I)$ يحقق الشروط التالية:

1. S مجموعة منتهية.
2. I جماعة غير خالية من المجموعات الجزئية من S، تسمى ***المجموعات الجزئية المستقلة*** ***independent subsets*** من S. بحيث أنه إذا كان $B \in I$ وكان $A \subseteq B$، فإن $A \in I$. ونقول إن I ***وراثية*** ***hereditary*** إذا حققت هذه الخاصية. لاحظ أن المجموعة الخالية $\emptyset$ هي بالضرورة عنصر في I.
3. إذا كان $A \in I$ و $B \in I$ و $|A| < |B|$ فثمة عنصر $x \in B - A$ بحيث $A \cup \{x\} \in I$. نقول أن M تحقق ***خاصية التبادل*** ***exchange property***.

يعود فضل ابتكار كلمة الكيان المصفوفي *"matroid"* إلى *Hassler Whitney*؛ فقد كان يدرس الكيان ***المصفوفي المصفوفاتي*** ***matric matroid***، الذي تكون فيه عناصر S أسطر مصفوفة معطاة وتكون مجموعة الأسطر مستقلة إذا كانت مستقلة خطيًّا بالمعنى المعتاد. هذه البنية تعرّف كيانًا مصفوفيًّا، يُطلب إليك في التمرين 16.4-2 بيان ذلك.

ثمة مثال آخر على الكيانات المصفوفية هو ***الكيان المصفوفي البياني*** ***graphic matroid*** $M_G = (S_G, I_G)$ المعرَّف بدلالة البيان غير الموجه $G = (V, E)$ كما يلي:

- المجموعة S_G هي المجموعة E، مجموعة الوصلات في G.
- إذا كانت A مجموعة جزئية من E، فإن $A \in I_G$ إذا وفقط إذا كانت A خالية من الحلقات. أي إن مجموعة الوصلات في A مستقلة إذا وفقط إذا كان البيان الجزئي $G_A = (V, A)$ يكوِّن غابة.

إن الكيان المصفوفي البياني M_G وثيق الصلة بمسألة شجرة المسح الصغرى، المشروحة بالتفصيل في الفصل 23.

مبرهنة 5.16

إذا كان $G = (V, E)$ بيانًا غير موجه، فإن $M_G = (S_G, I_G)$ كيانٌ مصفوفي.

البرهان من الواضح أنَّ $S_G = E$ مجموعة منتهية، وأنَّ I_G وراثية، لأن المجموعة الجزئية من غابة هي أيضًا غابة. وبطريقة أخرى، فإن حذف وصلات من مجموعة وصلات خالية من الحلقات لا يمكن أن ينشئ حلقات.

وهكذا فما علينا سوى أن نبيِّن أن M_G تحقِّق خاصية التبادل. لنفترض أن $G_A = (V, A)$ و $G_B = (V, B)$ غابتان في G وأن $|A| < |B|$. أي إن A و B هما مجموعتا وصلات خالية من الحلقات، و B تحتوي على وصلاتٍ أكثر من A.

إن الغابة $F = (V_F, E_F)$ تتضمن تمامًا $|V_F| - |E_F|$ شجرةً. ولبيان ذلك، نفترض أن F تتضمن t شجرةً، حيث للشجرة i v_i عقدة و e_i وصلة. فيكون لدينا

$$\begin{aligned}|E_F| &= \sum_{i=1}^{t} e_i \\ &= \sum_{i=1}^{t} (v_i - 1) \qquad \text{((اعتماداً على المبرهنة (ب.2))} \\ &= \sum_{i=1}^{t} v_i - t \\ &= |V_F| - t\ ,\end{aligned}$$

وهذا يقتضي أن $t = |V_F| - |E_F|$. وبذلك فإن الغابة G_A تتضمن $|V| - |A|$ شجرةً، والغابة G_B تتضمن $|V| - |B|$ شجرةً.

ولما كانت الغابة G_B تتضمن أشجارًا أقل من الغابة G_A، وَجَبَ أن تتضمن الغابة G_B شجرةً ما T عُقَدُها في شجرتين مختلفتين من الغابة G_A. يضاف إلى ذلك أنه لما كانت T مترابطة، وجب أن تتضمن وصلةً (u, v) بحيث تكون العقدتان u و v في شجرتين مختلفتين من الغابة G_A. ولما كانت الوصلة (u, v) تصل عقدتيْن من شجرتَيْن مختلفتين من الغابة G_A، فيمكننا إضافة الوصلة (u, v) إلى الغابة G_A دون إنشاء حلقة. وبهذا، تحقّق M_G خاصية التبادل، ويتمّ البرهان على أن M_G كيان مصفوفي. ■

إذا كان لدينا كيان مصفوفي $M(S, I)$، فإننا نسمي العنصر $x \notin A$ ***توسّع extension*** المجموعة $A \in I$ إذا أمكن إضافة x إلى A بحيث نحافظ على الاستقلال؛ أي إن x توسّع لـ A إذا كان $A \cup \{x\} \in I$. كمثال على ذلك، لنأخذ كيانًا مصفوفيًا بيانيًّا M_G؛ فإذا كانت A مجموعة مستقلة من الوصلات، فإن الوصلةَ e توسُّعٌ للمجموعة A إذا وفقط إذا لم تكن e من A وإذا لم تؤدِّ إضافةُ e إلى A إلى إنشاء حلقة.

إذا كانت A مجموعةً جزئيةً مستقلةً في كيان مصفوفي M، فإننا نقول عن A إنها ***عُظْمَى maximal*** إذا لم يكن لها أيُّ توسّع. أي تكون A عُظْمَى إذا لم تكن محتواة في أية مجموعة جزئية مستقلة من M أكبر منها. الخاصية التالية مفيدة في أحايين عديدة.

مبرهنة 6.16

كلُّ المجموعات الجزئية المستقلة العُظْمَى من كيانٍ مصفوفي لها الحجمُ نفسُه.

البرهان نفترض العكس؛ أي إن A مجموعةٌ جزئيةٌ مستقلةٌ عُظْمَى في M، وتوجد مجموعةٌ جزئيةٌ مستقلةٌ عُظْمَى أكبر منها B في M. وهذا يقتضي بموجب خاصية التبادل أنه يمكننا توسيع A إلى مجموعةٍ مستقلةٍ أكبر $A \cup \{x\}$، حيث $x \in B - A$. وهذا يناقض افتراض A عُظْمَى. ■

كتوضيح لهذه المبرهنة، لنأخذ كيانًا مصفوفيًّا بيانيًّا M_G لبيان مترابط غير موجه G. يجب أن تكون كلُّ

مجموعة جزئية مستقلة عُظْمَى في M_G شجرةً حرة لها $|V| - 1$ وصلةً تمامًا تصل جميع عقد G. نسمي مثل هذه الشجرة ***شجرة مسح spanning tree*** في G.

نقول إن الكيان المصفوفي $M = (S, I)$ ***مثقّل weighted*** إذا كان مرفَقًا بدالة ثقل (وزن) w تسند ثقلاً موجبًا تمامًا $w(x)$ لكل عنصر $x \in S$. تتوسع دالة الثقل w إلى المجموعات الجزئية في S بالجمع

$$w(A) = \sum_{x \in A} w(x)$$

لأية مجموعة جزئية $A \subseteq S$. على سبيل المثال، إذا جعلنا $w(e)$ تَرمز إلى ثقل الوصلة e في الكيان المصفوفي البياني M_G، فإن $w(A)$ هو الثقل الكلي للوصلات في مجموعة الوصلات A.

الخوارزميات الشرهة على كيان مصفوفي مثقَّل

يمكن صياغة العديد من المسائل -التي يعطي النهجُ الشرهُ حلولاً مُثْلَى لها- على أنها مسائل إيجاد مجموعة جزئية مستقلة بثقل أعظم في كيان مصفوفي مثقَّل. أي إنه يوجد كيانٌ مصفوفي مثقَّل $M = (S, I)$، ونود إيجاد مجموعةٍ مستقلة $A \in I$ بحيث يكون $w(A)$ ذا قيمة عُظْمَى. نسمي مثل هذه المجموعة الجزئية، التي هي مستقلة ولها ثقل أعظم ممكن، مجموعةً ***مُثْلَى optimal*** للكيان المصفوفي. ولما كان ثقْلُ أيِّ عنصرٍ $x \in S$ موجبًا، فإن أية مجموعة جزئية مُثْلَى هي دائمًا مجموعة جزئية مستقلة عُظْمَى- من المفيد دومًا جعل $w(A)$ كبيرة قدر الإمكان.

على سبيل المثال، في ***مسألة شجرة المسح الصغرى minimum-spanning-tree problem***، لدينا بيان مترابط غير موجه $G = (V, E)$، ودالةُ طولٍ w بحيث تكون $w(e)$ الطول (الموجب) للوصلة e. (نستخدم المصطلح "طول" هنا لنشير إلى أثقال الوصلة الأصلية في البيان، ونحتفظ بالمصطلح "ثقل" للإشارة إلى الأثقال في الكيان المصفوفي المرافق.) ونرغب في إيجاد مجموعةٍ جزئية من الوصلات التي تصل كل العقد معًا وبحيث يكون لها طولٌ كليٌّ أصغر. ولعرض هذه المسألة على أنها مسألة إيجاد مجموعة جزئية مُثْلَى في كيان مصفوفي، لنأخذ الكيان المصفوفي المثقّل M_G مع دالة الثقل w'، حيث $w'(e) = w_0 - w(e)$ و w_0 أكبر من الطول الأعظم لأيِّ وصلة. في هذا الكيان المصفوفي المثقّل، جميعُ الأثقال موجبة، والمجموعة الجزئية المُثْلَى هي شجرة مسح بطول كلي أصغر في البيان الأصلي. وبتعبير أدق، تُقابِل كلُّ مجموعةٍ جزئية مستقلة عُظْمَى A شجرةَ مسحٍ، لها $|V| - 1$ وصلة، ولما كان:

$$\begin{aligned} w'(A) &= \sum_{e \in A} w'(e) \\ &= \sum_{e \in A} (w_0 - w(e)) \end{aligned}$$

$$= (|V| - 1)w_0 - \sum_{e \in A} w(e)$$
$$= (|V| - 1)w_0 - w(A)$$

في حالة أية مجموعة جزئية مستقلة عُظْمَى A، فإن المجموعة الجزئية المستقلة التي تجعل الكمية $w'(A)$ عُظْمَى، عليها أن تجعل $w(A)$ صغرى. وبذلك، فإن أيَّ خوارزميةٍ توجِدُ مجموعةً جزئيةً مُثْلَى A في كيان مصفوفي اعتباطي يمكنها أن تَحُلَّ مسألةَ المسح الصغرى.

يَعرض الفصل 23 خوارزمياتٍ لمسألة شجرة المسح الصغرى، ولكننا نعرض هنا خوارزميةً شرهة، تصلح لأيِّ كيان مصفوفي مثقَّل. تأخذ الخوارزمية كيانًا مصفوفيًا $M = (S, I)$ على أنه دخلٌ لها، مع دالة ثقل موجب مرافق w ، وتعيد مجموعةً جزئيةً مُثْلَى A. نَرمز في شبه رمازنا إلى مكونات M بـ $M.S$ و $M.I$ ولدالة الثقل بـ w. إن هذه الخوارزمية شرهةٌ، لأنها تأخذ كلَّ عنصرٍ $x \in S$ بدوره في الترتيب المتناقص باطراد للوزن، وتضيفه مباشرةً إلى المجموعة A التي هي في قيد البناء إذا كانت المجموعة $A \cup \{x\}$ مستقلة.

```
GREEDY(M, w)
1  A = ∅
2  sort M.S into monotonically decreasing order by weight w
3  for each x ∈ M.S, taken in monotonically decreasing order by weight w(x)
4      if A ∪ {x} ∈ M.I
5          A = A ∪ {x}
6  return A
```

يَفحص السطر 4 ما يلي: هل تحافظ إضافةُ أيِّ عنصرٍ x إلى المجموعة A على استقلالية A؟ فإذا بقيت A مستقلةً، فإن السطر 5 يضيف x إلى A، وإلاَّ تُهمل x. ولما كانت المجموعةُ الحاليةُ مستقلةً، وكان كلُّ تكرارٍ في الحلقة **for** يحافظ على استقلالية A، فإن المجموعةَ الجزئيةَ مستقلةٌ دومًا (بالاستقراء). ولذلك، تعيد الخوارزميةُ GREEDY دائمًا مجموعةً مستقلة A. وسنجد بعد قليل أن A مجموعةٌ جزئيةٌ بثقل أعظم ممكن، وبذلك تكون A مجموعةً جزئيةً مُثْلَى.

من السهل تحليل زمن تنفيذ الخوارزمية GREEDY. لنرمز إلى $|S|$ بـ n. تَستغرق مرحلة الفرز في الخوارزمية GREEDY زمنًا $O(n \lg n)$. يُنفَّذ السطر 4، n مرة، مرة لكل عنصر من S. يتطلب كلُّ تنفيذٍ للسطر 4 فحْصَ ما يلي: هل المجموعة $A \cup \{x\}$ مستقلة أم لا؟ إذا استغرقت كلُّ عمليةِ فحصٍ زمنًا $O(f(n))$، فإن كاملَ الخوارزمية تنفَّذ بزمن $O(n \lg n + n f(n))$.

نبرهن الآن أن الخوارزميةَ GREEDY تعيد مجموعةً جزئيةً مُثْلَى.

توطئة 7.16 (الكيانات المصفوفية تتمتع بخاصية الخيار الشره)

لنفترض $M = (S, I)$ كيانًا مصفوفيًا مثقلاً، بدالة ثقل w ، وأن S مفروزة بحسب الترتيب المتناقص باطراد

للثقل. ليكن x أول عنصر في S بحيث تكون المجموعة $\{x\}$ مستقلة، إن وُجد. فإذا وُجد x، فتوجد مجموعةٌ جزئيةٌ مُثْلَى A من S تتضمن x.

البرهان إذا لم يكن مثل هذا العنصر x موجودًا، فإن المجموعةَ الجزئيةَ المستقلةَ الوحيدةَ تكون هي المجموعة الخالية، وتكون التوطئةُ صحيحةً بداهة. وإلا، فلتكن B مجموعةٌ جزئيةٌ مُثْلَى غير خالية. ونفترض أن $x \notin B$؛ وإلا فبجعل $A = B$ نحصل على مجموعة جزئية مُثْلَى من S تتضمن x.

لا يوجد عنصرٌ من B وزنُهُ أكبر من $w(x)$. ولبيان ذلك، نلاحظ أن $y \in B$ يقتضي أن تكون $\{y\}$ مستقلة، لأن $B \in I$ و I وراثية. وعلى ذلك، فإن خيارنا لـ x يَضمن أن يكون $w(x) \geq w(y)$ لأيِّ عنصر $y \in B$.

ننشئ المجموعة A كما يلي: نبدأ بـ $A = \{x\}$. تبعًا لطريقة اختيار x، تكون A مستقلة. وباستخدام خاصية التبادل، نوجد تكراريًّا عنصرًا جديدًا من B يمكن إضافته إلى A إلى أن يصبح $|A| = |B|$، مع الاحتفاظ باستقلالية A. عند هذه النقطة تكون A هي نفس B باستثناء أن A تتضمن x، و B تتضمن عنصرًا آخر y، أي إن $A = B - \{y\} \cup \{x\}$ لعنصر ما $y \in B$، وبذلك يكون:

$$
\begin{aligned}
w(A) &= w(B) - w(y) + w(x) \\
&\geq w(B) .
\end{aligned}
$$

ولما كانت المجموعة B مُثْلَى، وَجَبَ أن تكون المجموعة A – التي تتضمن x – مُثْلَى أيضًا. ■

سنبيّن لاحقًا أنه إذا لم يكن عنصرٌ ما خيارًا في البداية، فلن يكون خيارًا لاحقًا.

توطئة 8.16

ليكن $M = (S, I)$ أيَّ كيانٍ مصفوفي. إذا كان x عنصرًا من S وتوسعًا لمجموعة جزئية مستقلة A من S، فإن x توسّعٌ أيضًا للمجموعة الخالية $\emptyset$.

البرهان لما كان x توسّعًا لـ A، فإن $A \cup \{x\}$ مجموعة مستقلة. ولما كانت I وراثية، وَجَبَ أن تكون $\{x\}$ مستقلة، وبذلك فإن $\{x\}$ توسعٌ للمجموعة الخالية $\emptyset$. ■

نتيجة 9.16

ليكن $M = (S, I)$ أيَّ كيانٍ مصفوفي. إذا كان x عنصرًا من S بحيث لا يكون x توسعًا للمجموعة الخالية $\emptyset$، فإن x ليس توسعًا لأية مجموعة جزئية مستقلة A من S.

البرهان هذه النتيجة هي ببساطة الاقتضاء المعاكس الموجب contrapositive للتوطئة 8.16. ■

تعني النتيجة 9.16 أن أي عنصر إذا لم يكن بالإمكان استخدامه فورًا، فلن يُستخدَم أبدًا. لذلك، لا يمكن أن تخطئ خوارزمية GREEDY بترك أي عناصر بدئية في S ليست توسعًا لـ $\emptyset$ جانبًا، لأنها لن تُستخدَم أبدًا.

توطئة 10.16 (الكيانات المصفوفية تتمتع بخاصية البنية الجزئية المُثلَى)

ليكن x أول عنصر في S تختاره الخوارزمية GREEDY للكيان المصفوفي المثقّل $M = (S, I)$. المسألة المتبقية وهي إيجاد مجموعة جزئية مستقلة ذات وزن أعظم تتضمن x، تُختَصَر إلى مسألة إيجاد مجموعة جزئية مستقلة ذات وزن أعظم من الكيان المصفوفي المثقل $M' = (S', I')$، حيث

$$S' = \{y \in S : \{x, y\} \in I\} ,$$
$$I' = \{B \subseteq S - \{x\} : B \cup \{x\} \in I\} ,$$

ودالة الثقل لـ M' هي دالة الثقل لـ M، مقصورةً على S'. (نسمي M' ***تقليص*** ***M contraction*** اعتمادًا على العنصر x.)

البرهان إذا كانت A أية مجموعة جزئية مستقلة ذات وزن أعظم من M تتضمن x، عندها تكون $A' = A - \{x\}$ مجموعة جزئية مستقلة في M'. وبالعكس أية مجموعة جزئية مستقلة A' من M' تعطي مجموعةً جزئية مستقلة $A = A' \cup \{x\}$ من M. ولما كان لدينا في كلتا الحالتين $w(A) = w(A') + w(x)$، فإن الحل ذا الوزن الأعظم في M الذي يتضمن x يؤدي إلى حل ذي ثقل أعظم في M' وبالعكس. ■

مبرهنة 11.16 (صحة الخوارزمية الشرهة على الكيانات المصفوفية)

إذا كان $M = (S, I)$ كيانًا مصفوفيًّا مثقلاً بدالة ثقل w، فإن الإجراء $\text{GREEDY}(M, w)$ يعيد مجموعة جزئية مُثلَى.

البرهان نستنتج من النتيجة 9.16 أن كل العناصر التي يُهملها GREEDY منذ البداية (لأنها ليست توسعًا لـ $\emptyset$) يمكن تجاهلها، لأنها لن تكون مفيدة. وحين يختار GREEDY العنصرَ الأول x، فإن التوطئة 7.16 تقتضي أن الخوارزمية لا تخطئ بإضافة x إلى A، لوجود مجموعة جزئية مُثلَى تتضمن x. أخيرًا، ينتج عن التوطئة 10.16 أن المسألة المتبقية هي مسألة إيجاد مجموعة جزئية مُثلَى في الكيان المصفوفي M'، وهو تقليصُ M اعتمادًا على x. وبعد أن يُعطيَ الإجراءُ GREEDY المجموعةَ A القيمةَ $\{x\}$، فإنه يمكننا تفسير جميع الخطوات المتبقية على أنها تعمل في الكيان المصفوفي $M' = (S', I')$، لأن المجموعة B تكون مستقلة في M' إذا وفقط إذا كانت $B \cup \{x\}$ مستقلة في M، لكل المجموعات $B \in I'$. وبذلك، ستجد العمليةُ التاليةُ من GREEDY مجموعةً جزئيةً مستقلةً ذات ثقل أعظم على M'، وسينتج عن مجمل الإجراء Greedy مجموعة جزئية مستقلة مُثلَى في M. ■

تمارين

1-4.16

بيّن أن (S, I_k) هو كيان مصفوفي، حيث S مجموعة منتهية و I_k مجموعة كل المجموعات الجزئية من S التي حجمها k على الأكثر، حيث $k \le |S|$.

2-4.16 ★

لتكن لدينا المصفوفة T، ذات البعدين $m \times n$، المعرّفة على حقل ما (مثل حقل الأعداد الحقيقية). بيّن أن (S, I) هو كيان مصفوفي، حيث S مجموعة الأعمدة في T و $A \in I$ إذا وفقط إذا كانت أعمدة A مستقلة خطيًّا.

3-4.16 ★

بيّن أنه إذا كان (S, I) كيانًا مصفوفيًّا، فإن (S, I') كيان مصفوفي، حيث:

$I' = \{A' : S - A' \text{ contains some maximal } A \in I\}$.

أي إن المجموعات المستقلة العُظْمَى في (S, I') هي متممات المجموعات المستقلة العُظْمَى في (S, I).

4-4.16 ★

لتكن S مجموعة منتهية، ولتكن $S_1, S_2, \dots, S_k$ تجزئة S إلى مجموعات جزئية منفصلة غير خالية. عرِّف البنية (S, I) بشرط $I = \{A : |A \cap S_i| \le 1 \text{ for } i = 1, 2, \dots, k\}$. بيّن أن (S, I) كيان مصفوفي؛ أي إن مجموعة كل المجموعات A التي تتضمن عضوًا (عنصرًا) واحدًا على الأكثر في كل كتلة من التجزئة، تحدِّد المجموعات المستقلة للكيان المصفوفي.

5-4.16 ★

بيّن كيف تحوِّل دالةَ الثقل لمسألة كيان مصفوفي مثقّل، حيث الحل الأمثل المرغوب هو مجموعة جزئية مستقلة عُظْمَى ذات ثقل أصغر، لجعله مسألة كيان مصفوفي مثقّل معياري. ناقش بعناية صحة تحويلك.

★ 5.16 مسألة جدولة المهام

من المسائل المهمة التي يمكن حلها باستخدام الكيانات المصفوفية مسألة جدولة مهام في واحدة الزمن، جدولة مُثْلَى، باستخدام معالج واحد، حيث لكل مهمة حدّ انتهاء محدد، يجب بعده دفع غرامة إذا لم تُنفَّذ المهمة قبل حدِّ انتهائها. تبدو المسألة معقدة، إلا أنه يمكننا حلها بطريقة غاية في البساطة وذلك بتحويلها إلى كيان مصفوفي واستخدام خوارزمية شرهة.

المهمة في واحدة الزمن ***unit-time task*** هي عملٌ، مثل برنامج، يجب تنفيذه على حاسوب ويتطلب

تمامًا واحدةَ زمنٍ لاستكماله. إذا كانت لدينا مجموعة منتهية S من المهام في واحدة الزمن، فإن ***جدولة schedule*** المجموعة S هي تبديل على S يحدد الترتيب الذي يجب إنجاز هذه المهام وفقه. تبدأ المهمة الأولى في الجدولة في اللحظة 0 وتنتهي في اللحظة 1، وتبدأ المهمة الثانية في اللحظة 1 وتنتهي في اللحظة 2، وهكذا...

لمسألة ***جدولة المهام في واحدة الزمن، بحدود انتهاء وغرامات، على معالج واحد scheduling unit time tasks with deadlines and penalties for a single processor***، المدخلات التالية:

- مجموعة $S = \{a_1, a_2, ..., a_n\}$ من n مهمة في واحدة الزمن.
- مجموعة من n عددًا طبيعيًّا هي ***حدود انتهاء deadlines*** $d_1, d_2, ..., d_n$ بحيث تحقق كلُّ d_i: $1 \leq d_i \leq n$، ويُفترَض أن تنتهي المهمة a_i قبل اللحظة d_i: و
- مجموعة من n ثقلاً غير سالب، أو ***غرامة penalty*** $w_1, w_2, ..., w_n$، بحيث نتكبّد الغرامة w_i إذا لم تنته المهمة a_i بحلول d_i، ولا نتعرض لأية غرامة إذا انتهت المهمة قبل حلول حدّ انتهائها.

نرغب في إيجاد جدول لـ S يصغّر الغرامة الكلية الناتجة عن تجاوز حدود الانتهاء.

لنأخذ جدولاً معينًا. نقول عن مهمة إنها ***متأخرة late*** في هذا الجدول إذا انتهت بعد حدِّ انتهائها، وإلا فنقول إن هذه المهمة ***مبكرة early*** في الجدول. يمكن دائمًا وضع أي جدول اعتباطي ***بالشكل المبكر أولاً early first form***. وفي هذا الشكل تسبق المهامُّ المبكرة المهامَّ المتأخرة. ولبيان ذلك، لاحظ أنه إذا لحقت مهمةٌ مبكرة a_i مهمةً متأخرة a_j، عندها يمكننا المبادلة بين a_i و a_j، وتبقى a_i مبكرة و a_j متأخرة.

يضاف إلى ذلك، أنه يمكن دومًا تحويل أي جدول اعتباطي إلى ***الشكل القانوني canonical form***، الذي تَسبق فيها المهامُّ المبكرة المهامَّ المتأخرة، وتُجدوَل المهامُّ المبكرة بالترتيب المتزايد باطّراد لحدود انتهائها. وللقيام بذلك، نضع الجدول بالشكل المبكر أولاً. ثم إذا وجدنا مهمتَيْن مبكرتين a_i و a_j تنتهيان في الجدول في اللحظات k و $k+1$ على الترتيب بحيث يكون $d_j < d_i$، فإننا نبدل مواضع a_i و a_j. ولما كانت a_j مبكرة قبل الإبدال، فإن $k + 1 \leq d_j$. إذن، $k + 1 < d_i$، وبذلك تبقى a_i مبكرة بعد الإبدال. وبسبب تحريك المهمة a_j إلى موضع أبكر في الجدول، فإنها تبقى مبكرة بعد الإبدال.

وهكذا يؤول البحثُ عن الجدول الأمثل إلى إيجاد مجموعة مهام A التي يجب أن تكون مبكرة في الجدول الأمثل. فإذا حدّدنا A، أمكننا إنشاء الجدول الفعلي بسرد عناصر A بالترتيب المتزايد باطراد لحدود الانتهاء، ثم بسرد المهام المتأخرة (أي $S - A$) بأي ترتيب، وهذا يُنتِج ترتيبًا قانونيًّا للجدول الأمثل.

نقول عن مجموعة من المهام A إنها ***مستقلة independent*** إذا وُجد جدول لهذه المهام لا يكون فيها أية مهمة متأخرة. من الواضح أن مجموعةَ المهام المبكرة في أي جدول تكوّن مجموعةً مستقلةً من المهام. لِنرمز بـ I إلى مجموعة كل المجموعات المستقلة من المهام.

ندرس فيما يلي مسألة الحكم على مجموعةٍ معطاةٍ من المهام A بأنها مستقلة. نرمز بـ $N_t(A)$ إلى عدد المهام في A التي لا يتجاوز حد انتهائها t حيث $t = 0, 1, \dots, n$. لاحظ أن $N_0(A) = 0$ مهما كانت المجموعة A.

توطئة 12.16

مهما كانت مجموعة المهام A، فإن العبارات التالية متكافئة

1. المجموعة A مستقلة.
2. $N_t(A) \leq t$، حيث $t = 0, 1, \dots, n$.
3. إذا جُدولت المهام في A بالترتيب المتزايد باطراد لحدود الانتهاء، فلن توجد أية مهمة متأخرة.

البرهان سنُثبت أن (1) تقتضي (2) بطريقة الاقتضاء المعاكس الموجب. إذا كان $N_t(A) > t$ للحظة ما t، فلا توجد طريقة لبناء جدول لا يتضمن أي مهام متأخرة في المجموعة A، بسبب وجود أكثر من t مهمةً يجب أن تنتهي قبل t. لذلك فإن (1) تقتضي (2). فإذا كانت (2) صحيحة، وَجَبَ أن تكون (3) صحيحةً بالضرورة؛ إذ لا مجال للوقوع في "مأزق" عند جَدْولة المهام بالترتيب المتزايد باطراد لحدود الانتهاء، لأن (2) تقتضي أن حد النهاية ذا الترتيب i من حيث كبر القيمة يساوي i على الأكثر. أخيرًا، (3) تقتضي بداهة (1). ■

باستخدام الخاصية 2 من التوطئة 12.16، يمكننا بسهولة حساب كون مجموعة مهام معطاة A مستقلة أم لا (انظر التمرين 16.5-2).

إن مسألةَ تصغيرِ minimizing مجموع الغرامات عن المهام المتأخرة هي نفسها مسألة تعظيم maximizing مجموع الغرامات عن المهام المبكرة. ولهذا، فإن المبرهنةَ التالية تؤكد أنه يمكننا استخدام الخوارزميات الشرهة لإيجاد مجموعة A من المهام المستقلة بحيث تكون الغرامة الكلية عُظْمَى.

مبرهنة 13.16

إذا كانت S مجموعة مهام في واحدة الزمن مع حدود انتهاء، وكانت I مجموعة كل مجموعات المهام المستقلة، فإن النظام الموافق (S, I) هو كيانٌ مصفوفي.

البرهان إنَّ كلَّ مجموعةٍ جزئيةٍ من مجموعةِ مهامَّ مستقلةٍ، هي مستقلةٌ بالتأكيد. لبرهان خاصية التبادل، نفترض أن B و A مجموعتان مستقلتان من المهام، وأن $|B| > |A|$. ولتكن k أكبر t بحيث يكون $N_t(B) \leq N_t(A)$، (إن هذه القيمة موجودة، لأن $N_0(A) = N_0(B)$). ولما كان $N_n(B) = |B|$ و $N_n(A) = |A|$، و $|B| > |A|$، فيجب أن يكون $k < n$ و $N_j(B) > N_j(A)$ لكل قيم j في

مهمة

a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

الشكل 7.16 منتسَخ (مثال) عن مسألة جدولة مهام في واحدة الزمن، مع حدود انتهاء وغرامات لمعالِج واحد.

المجال $k+1 \le j \le n$. لذا، فإن B تتضمن مهامَّ بحدودِ انتهاءٍ تساوي $k+1$ أكثر مما تتضمنه A. لتكن a_i مهمة في $B - A$ مع حدِّ انتهاء $k+1$، ولتكن $A' = A \cup \{a_i\}$.

نبيّن الآن أن A' يجب أن تكون مستقلة باستخدام الخاصية 2 من التوطئة 12.16. لدينا $N_t(A') = N_t(A) \le t$ مهما كانت $0 \le t \le k$، لأن A مستقلة. ولدينا $N_t(A') \le N_t(B) \le t$ مهما كانت $k < t \le n$، لأن B مستقلة. لذا، فإن A' مستقلة، وهذا يتمّم برهاننا أن (S, I) كيان مصفوفي. ■

يمكننا اعتمادًا على المبرهنة 11.16 استخدامُ خوارزميةٍ شرهة، لإيجاد مجموعةِ مهامَّ مستقلةٍ ذات وزن أعظم. يمكننا بعدها إنشاءُ جدولٍ أمثلَ تكون مهامُّ A هي مهامَّهُ المبكرة. تُعَدُّ هذه الطريقةُ خوارزميةً فعالةً لجدولةِ مهامَّ في واحدة الزمن مع حدود انتهاء وغرامات لمعالِجٍ واحد. إن زمن التنفيذ هو $O(n^2)$ باستخدام GREEDY، لأن كلَّ فحصٍ من فحوص الاستقلالية الـ $O(n)$ التي تقوم بها الخوارزمية يستغرق زمنًا $O(n)$ (انظر التمرين 16.5-2). ثمة خوارزميةٌ أسرعُ إنجازًا نجدها في المسألة 16-4.

يبيّن الشكل 7.16 مثالاً على مسألة جدولة مهام في واحدة الزمن، مع حدود انتهاء وغرامات لمعالِج واحد. في هذا المثال، تختار الخوارزمية الشرهة المهام a_1 و a_2 و a_3 و a_4، على الترتيب، ثم ترفض كلاًّ من a_5 (لأن $N_4(\{a_1, a_2, a_3, a_4, a_5\}) = 5$) و a_6 (لأن $N_4(\{a_1, a_2, a_3, a_4, a_6\}) = 5$)، وأخيرًا تقبل a_7. ويكون الجدول الأمثل النهائي هو:

$$\langle a_2, a_4, a_1, a_3, a_7, a_5, a_6 \rangle ,$$

والغرامة الكلية هي $w_5 + w_6 = 50$.

تمارين

16.5-1

حُلَّ مثالَ مسألةِ الجدولة المعطاة في الشكل 7.16 ولكن بإبدال الغرامة w_i بـ $80 - w_i$.

16.5-2

بيّن كيف تُستَخدم الخاصية 2 من التوطئة 12.16 لتحديد كون مجموعة من المهام A مستقلةً أم لا في زمن $O(|A|)$.

مسائل

1-16 *تبديل العملات*

ادرس مسألة صرف n سنتًا باستخدام أقل عدد من القطع النقدية. افترض أن قيمة كل قطعة نقدية هي عدد صحيح.

أ. وَصِّفْ خوارزميةً شرهةً للصرف مكوَّنةً من: أرباع الدولار، وأعشارِه dimes، ونِكْلاتِه nickels (النِّكْلَة تساوي خمسة سَنْتات)، وسَنْتاتِه pennies (السَّنْت يساوي 1/100 من الدولار). برهن أن خوارزميتك تحقق حلاً أمثل.

ب. افترضْ أن كسورَ مقامات قطع النقود المتاحة (الفئات النقدية) هي قوًى صحيحةٌ لـ c، أي إن المقامات هي $c^0, c^1, \ldots, c^k$ حيث $c > 1$ و $k \geq 1$ عددان صحيحان. بيّن أن الخوارزمية الشرهة تحقّق دائمًا حلاً أمثلَ.

ت. أعْطِ مجموعةً من الفئات النقدية تجعل الخوارزميةَ الشرهة لا تعطي حلاً أمثل. يجب أن تتضمن مجموعتك سَنْتًا بحيث يوجد حلٌ لكلّ قيمة n.

ث. أعْطِ خوارزميةً بزمن $O(nk)$ تحقِّق عملية الصرف لأية مجموعة مؤلَّفةٍ من k قطعة نقدية مختلفة، بافتراض أن السَّنْتَ أحدُ هذه القطع.

2-16 *جدولة لتصغير متوسط لحظات الإنهاء*

لنفترض أن لديك مجموعة من المهام $S = \{a_1, a_2, \ldots, a_n\}$، حيث تتطلب المهمة a_i، حين تبدأ، عددًا p_i من وحدات المعالجة لإنهائها. ولديك حاسوبٌ واحدٌ لتنفيذ هذه المهام عليه. يستطيع هذا الحاسوب تنفيذَ مهمةٍ واحدة في لحظة معينة. ولتكن c_i ***لحظة إنهاء completion time*** المهمة a_i، أي اللحظة التي تنتهي عندها معالجة المهمة a_i. والمطلوب هو تصغير متوسط زمن الإنهاء، أي تصغير $(1/n)\sum_{i=1}^{n} c_i$. افترض، على سبيل المثال، أن لديك مهمتين a_1 و a_2، وأن $p_1 = 3$ و $p_2 = 5$، ولنأخذ الجدول الذي تُنفَّذ فيه a_2 أولاً تتبعها a_1. عندها يكون $c_2 = 5$ و $c_1 = 8$، ومتوسط لحظات الإنهاء $(5 + 8)/2 = 6.5$. وإذا نُفِّذت المهمةُ a_1 أولاً، فإن $c_1 = 3$ و $c_2 = 8$، ومتوسط لحظات الإنهاء $(3 + 8)/2 = 5.5$.

أ. أعْطِ خوارزميةً تجدول المهام بحيث تصغِّر متوسط لحظات الإنهاء. يجب أن تُنفَّذ كلُّ مهمةٍ دون استحواذ، أي حين تبدأ المهمة a_i يجب أن تنفَّذ باستمرار خلال p_i وحدة زمن. أثبت أن خوارزميتك تصغِّر متوسط لحظات الإنهاء، وأعطِ زمن تنفيذ خوارزميتك.

ب. افترض الآن أن المهام ليست جميعها متاحة معًا. أي إن كلَّ مهمةٍ لا يمكن أن تبدأ ما لَم تَحِنْ ***لحظة***

إصدارها release time r_i. افترض أيضًا أننا نسمح ***بالاستحواذ preemption***، بحيث يمكن تعليق مهمة ثم إعادة تشغيلها في وقت لاحق. فمثلاً، يمكن لمهمةٍ a_i (زمنُ معالجتها $p_i = 6$، وزمنُ إصدارها $r_i = 1$) أن تبدأ بالتنفيذ في اللحظة 1 ثم يُستحوَذ عليها في اللحظة 4. يمكن استئنافها في اللحظة 10 والاستحواذ عليها في اللحظة 11، وأخيرًا استئنافها في اللحظة 13 وإنهاؤها في اللحظة 15. جرى تنفيذ المهمة a_i خلال زمن كلي يساوي 6 وحدات زمن، إلا أن زمن تنفيذها قد قُسم إلى ثلاث قطع. نقول إن لحظة إنهاء a_i هو 15. أعطِ خوارزميةً تجدول المهام بحيث تصغِّر متوسطَ لحظات الإنهاء في هذا السيناريو. برهن أن خوارزميتك تصغِّر متوسط لحظات الإنهاء، وأعطِ زمن تنفيذها.

3-16 البيانات الجزئية الخالية من الحلقات

أ. ***مصفوفة الورود incidence matrix*** لبيانٍ غير موجَّه $G = (V, E)$ هي مصفوفة M $|V| \times |E|$ حيث $M_{ve} = 1$ إذا كانت الوصلة e واردة على العقدة v، وإلا فإن $M_{ve} = 0$. ناقش ما يلي: تكون مجموعةٌ من أعمدة M مستقلةً خطيًّا على حقل الأعداد الصحيحة بالمقاس 2، إذا وفقط إذا كانت مجموعةُ الوصلات الموافقة خاليةً من الحلقات.

ب. لنفترض أننا نرفق ثقلاً غير سالب $w(e)$ بكل وصلةٍ من بيانٍ غير موجَّه $G = (V, E)$. أعْطِ خوارزميةً فعالة لإيجاد مجموعة جزئية من E خالية من الحلقات وبحيث يكون ثقلها الكلي أعظميًّا.

ت. ليكن $G = (V, E)$ بيانًا موجَّهًا، وليكن (E, I) معرَّفًا بحيث يكون $A \in I$ إذا وفقط إذا لم يتضمن A أية حلقات موجهة. أعْطِ مثالاً على بيانٍ موجه G بحيث لا يكون النظام الموافق (E, I) كيانًا مصفوفيًّا. حدِّد الشرطَ غير المحقَّق من تعريف الكيان المصفوفي.

ث. إذا كانت ***مصفوفة الورود incidence matrix*** لبيانٍ موجَّه $G = (V, E)$ من دون حلقات ذاتية هي مصفوفة $|V| \times |E|$ بحيث يكون $M_{ve} = -1$ إذا كانت الوصلة e تغادر العقدة v و $M_{ve} = 1$ إذا كانت الوصلة e تدخل العقدة v، و $M_{ve} = 0$ فيما عدا ذلك. ناقش ما يلي: إذا كانت مجموعة جزئية من أعمدة M مستقلةً خطيًّا، فإن مجموعةَ الوصلاتِ الموافقةَ لا تتضمن حلقةً موجَّهة.

ج. يقرِّر التمرين 2-4.16 أن مجموعةَ مجموعات الأعمدة المستقلة خطيًّا لأية مصفوفة M تكوِّن كيانًا مصفوفيًّا. فسِّر بعناية لماذا لا تتناقض نتائج الجزأين (ت) و (ث) فيما بينها. كيف يمكن ألا يتحقق التوافق الكامل بين مفهوم الوصلات الخالية من الحلقات ومفهوم الاستقلال الخطي لمجموعة الأعمدة الموافقة من مصفوفة الورود.

4-16 أنماط أخرى من مسألة الجدولة

لندرس الخوارزميةَ التالية للمسألة المذكورة في المقطع 5.16، وهي خاصة بجدولة مهام في واحدة الزمن مع حدود

انتهاء وغرامات. لتكن جميع الشرائح الزمنية (عددها n) فارغة في البداية، حيث الشريحة الزمنية i هي الشريحة التي طولها واحدة الزمن وتنتهي في اللحظة i. نأخذ المهام مرتبةً بحسب الغرامات المتناقصة باطراد. ندرس المهمة a_j كما يلي: إذا وُجد للمهمة a_j شريحةٌ زمنية عند حدِّ انتهاء d_j (أو قبله) وما تزال فارغة، فإننا نُسند a_j إلى آخر شريحة تحقق ذلك، فتصبح الشريحة ملأى (أو مشغولة). وإذا لم نجد هذه الشريحة الزمنية، فإننا نُسند a_j إلى آخر شريحة لم تُملأ بَعْدُ.

أ. أثبت أن هذه الخوارزمية تعطي دائمًا حلاً أمثلَ.

ب. استخدم غابة المجموعات المنفصلة السريعة المشروحة في المقطع 3.12 لتنجيز الخوارزمية بفعالية. افترض أن مجموعة مهام الدخل، مرتبة سابقًا بحسب الترتيب المتناقص باطراد للغرامات. حلّل زمن تنفيذ خوارزميتك.

5-16 *التخبئة المفصولة عن الخط*

تَستخدم الحواسيبُ الحديثة خابيةً لخزن كمياتٍ صغيرة من المعطيات في ذاكرة سريعة. ومع أن برنامجًا ما قد يَنْفُذ إلى كمياتٍ كبيرة من المعطيات، إلا أن خزْنَ مجموعةٍ جزئية صغيرة من الذاكرة الرئيسة في ***الخابية cache*** – وهي ذاكرة صغيرة ولكنها أسرع – يمكن أن يخفِّض زمن النفاذ الكلي تخفيضًا كبيرًا. حين يُنفَّذ برنامجٌ حاسوبي ما فإنه يَصنع متتاليةً $\langle r_1, r_2, \ldots, r_n \rangle$ من طلباتِ النفاذ إلى الذاكرة، حيث يكون كل طلبٍ لعنصر معطيات خاص. على سبيل المثال، يمكن لبرنامج يَنْفُذُ إلى أربعة عناصر متمايزة $\{a, b, c, d\}$ أن يَصنع متتالية الطلبات $\langle d, b, d, b, d, a, c, d, b, a, c, b \rangle$. ليكن k حجم الخابية. حين تتضمن الخابيةُ k عنصرًا، ويطلب البرنامجُ العنصرَ ذا الترتيب $k + 1$، فيجب أن يقرِّر النظامُ، لهذا الطلب ولكلِّ طلبٍ تالٍ، ما هي العناصر الـ k التي يجب أن يحتفظ بها في الخابية. وبعبارةٍ أدقّ، تتحقَّق خوارزميةُ إدارة الخابية من وجود العنصر r_i سابقًا في الذاكرة، وذلك لكل طلبٍ r_i. فإذا كان موجودًا فلدينا ***إصابة الخابية cache-hit***، وإلا فلدينا ***عدم إصابة الخابية cache-miss***. في حالة عدم الإصابة، يسحب النظام العنصر r_i من الذاكرة الرئيسية، وعلى خوارزمية إدارة الخابية أن تقرِّرَ إما الاحتفاظَ بالعنصر r_i في الخابية وإما عدم الاحتفاظ. فإذا قرَّرتِ الاحتفاظَ بالعنصر r_i، وكان في الخابية k عنصرًا، وجب أن تطرد عنصرًا لتخلي مكانًا لـ r_i. تطرد خوارزمية إدارة الخابية عناصر بهدف تصغير عدد مرات عدم إصابة الخابية على كامل متتالية الطلبات.

إن مسألة الخابية هي عادة مسألة على الخط on-line. أي علينا أن نتخذ قرارًا بشأن المعطيات التي ينبغي أن تبقى في الخابية دون أن نعلم الطلبات المستقبلية. ولكننا هنا ندرس النسخة المفصولة عن الخط off-line من هذه الخوارزمية، حيث لدينا سلفًا متتالية الطلبات كاملةً (وعددها n طلبًا) وحجم الخابية k، ونريد أن نجعل العددَ الكليَّ لحالات عدم الإصابة أصغريًّا.

يمكننا حل هذه المسألة المفصولة عن الخط باستخدام استراتيجية شرهة تسمى ***الأبعد في المستقبل*** ***further-in-future***، حيث تختار طرد البند الموجود في الخابية الذي يكون نفاذه القادم في متتالية طلبات النفاذ الأبعد في المستقبل.

أ. اكتب شبه رماز لمدير خابية يستخدم استراتيجية الأبعد في المستقبل. ينبغي أن يكون الدخل متتالية الطلبات $\langle r_1, r_2, \ldots, r_n \rangle$ وحجم الخابية k، وينبغي أن يكون الخرج متتالية القرارات المتعلقة بعناصر المعطيات التي يجب إخلاؤها عند كل طلب (إن وجدت). ما هو زمن تنفيذ هذه الخوارزمية؟

ب. بيّن أن مسألة التخبئة المفصولة عن الخط تُظهِر بنيةً جزئيةً مُثْلَى.

ت. بيّن أن إجراء الأبعد في المستقبل يُنتِج أصغر عددٍ ممكن من حالات عدم إصابة الخابية.

ملاحظات الفصل

يمكن أن نجد المزيدَ عن الخوارزميات الشرهة والكيانات المصفوفية في Lawler [224] و Papadimitriou و Steiglitz [271].

ظهرت الخوارزمية الشرهة أول مرة في كتب الأمثَلَة التوافقية في عام 1971 في مقالٍ لـ Edmonds [101]، علمًا بأن نظرية الكيانات المصفوفية تعود إلى تاريخ سابق في عام 1935 في مقالٍ لـ Whitney [355].

يعتمد برهاننا على صحة الخوارزمية الشرهة في مسألة احتيار النشاطات على المرجع Gavril [131]. ونجد دراسةَ مسألةِ جدولة المهام في Lawler [224]؛ Horowitz و Sahni و Rajasekaran [181] و Brassard و Bratley [55].

ابتُكرت أرمزةُ Huffman في عام 1952 [185]، وقد استكشف Lelewer و Hirschberg [231] تقنياتِ ضغط المعطيات التي كانت معروفة حتى عام 1987.

أما توسيع نطاق نظرية الكيان المصفوفي إلى نظرية الكيان الشره، فقد كان رائداها Korte و Lovász [216, 217, 218, 219]، اللذين عمَّما النظرية التي قدمناها هنا.

17 تحليل الكلفة المخمّدة

عند إجراء تحليل ***الكلفة المخمّدة amortized analysis***، نحسب متوسط الزمن اللازم للقيام بمتتالية من العمليات على بنية معطيات ما. يمكن استخدام هذا التحليل لبيان أن متوسط كلفة عملية ما صغير عندما نقوم بحساب المتوسط على متتالية من العمليات، حتى لو كانت عملية واحدة من بين هذه العمليات مكلفة. يختلف تحليل الكلفة المخمّدة عن تحليل الحالة الوسطى في أنه لا يستخدم الاحتمالات؛ فتحليل الكلفة المخمّدة يعطي ضمانة على *الأداء المتوسط لكل عملية في أسوأ الحالات*.

تعالج المقاطعُ الثلاثة الأولى من هذا الفصل التقنيات الثلاث الأكثر انتشارًا في تحليل الكلفة المخمّدة. إذ يبدأ المقطع 1.17 بالتحليل المجمَّع aggregate analysis، الذي نحدّد باستخدامه حدًّا أعلى $T(n)$ للكلفة الكليّة لمتتالية من n عملية. وبذلك يكون متوسط كلفة العملية الواحدة هو $T(n)/n$. ونعتبر أن الكلفة المتوسطة هي الكلفة المخمدة لكل عملية، وبهذا يكون لكل العمليات الكلفة المخمّدة نفسها.

ويتناول المقطعُ 2.17 طريقةَ المحاسبة، التي نحدّد فيها كلفة مخمّدة لكل عملية، وعندما يكون هناك أكثر من نمط من العمليات، يكون لكل نمط كلفة مخمدة مختلفة. تحمِّل طريقةُ المحاسبة بعضَ العمليات في بداية متتالية العمليات كلفةً إضافية، وتخزِّن هذه الكلفة الإضافية كـ "رصيد مسبق الدفع" لغرضٍ محدّد في بنية المعطيات. يُستخدم هذا الرصيد لاحقًا للتسديد عن عمليات بأن يُطلب منها أقل مما تكلّف فعلاً.

يناقش المقطع 3.17 طريقة الكمون، التي تحدِّد – بطريقةٍ مشابهة لطريقة المحاسبة – الكلفةَ المخمّدة لكل عملية، وقد نحمِّل بعضَ العمليات المبكرة أكثر من كلفتها لنعوّض عن تحميل عمليات بأقل من الكلفة لاحقًا. تحتفظ طريقة الكمون بقيمة اعتماد على أنها "الطاقة الكامنة" لبنية المعطيات ككل، بدلاً من ربط الاعتماد بأغراض فردية داخل بنية المعطيات.

سنستخدم مثالين ندرس من خلالهما هذه الطرق الثلاث. الأول هو مكدس مع عملية إضافية، هي MULTIPOP، التي تنْزع عدّة أغراض من المكدس دفعةً واحدة. والمثال الثاني هو عداد اثناني يعدّ بدءًا من 0 بالاعتماد على عملية وحيدة هي INCREMENT.

تذكّر، وأنت تقرأ هذا الفصل، أن الحمولات المسندة خلال عملية تحليل الكلفة المخمّدة ما هي إلا بهدف التحليل فقط، ولا ضرورة لظهورها في الرماز، بل ينبغي ألاّ تظهر فيه. فمثلاً، إذا أُسند اعتمادٌ إلى

غرضٍ ما x عند استخدام طريقة المحاسبة، فليست هناكَ حاجة إلى إسناد مقدار مناسب إلى واصفة ما – مثل $x.credit$ – في الرماز.

عندما نقوم بتحليل الكلفة المخمّدة لبنية معطيات محددة، فإننا غالبًا ما نكتسب نظرة أعمق قد تساعد في تحسين التصميم. وسنستخدم مثلاً في المقطع 4.17، طريقة الكمون لنحلّل جدولاً يتمدّد ويتقلّص ديناميكيًا.

1.17 التحليل المجَمَّع

نبيّن، في ***التحليل المجَمَّع aggregate analysis***، أن متتاليةً من n عملية تستغرق زمنًا كليًا *في أسوأ الحالات* هو $T(n)$، لكل قيم n. إن متوسط كلفة العملية الواحدة، أو ***كلفتها المخمّدة amortized cost***، في أسوأ الحالات هو $T(n)/n$. لاحظ أن هذه الكلفة المخمدة تنطبق على كلّ عملية، حتى عندما يكون هناك عدة أنواع من العمليات في المتتالية. أما فيما يخص الطريقتين التاليتين اللتين سندرسهما في هذا الفصل – طريقة المحاسبة وطريقة الكمون – فإنهما قد تسنِدان كلفًا مخمدة مختلفة إلى الأنواع المختلفة من العمليات.

عمليات المكدس

ندرس في مثالنا الأول المتعلق بالتحليل المجَمَّع كدساتٍ أضفنا إليها عملية جديدة. عرضنا في المقطع 1.10 عمليتَي المكدس الأساسيتين، وذكرنا أن كلاً منهما يستغرق زمنًا $O(1)$، وهما:

PUSH(S,x) تدفع الغرض x إلى المكدس S.

POP(S) تنزِع الغرض من أعلى المكدس S، وتعيده. وإن استدعاء عملية POP على مكدسٍ فارغ يُحدِث خطأً.

ولما كانت كلتا العمليتين تُنفَّذان في زمن $O(1)$، فسنفترض أن كلفة كلّ منها هي 1. وبذلك تكون الكلفة الكلية لمتتالية من n عملية PUSH وPOP هي n، ويكون زمن التنفيذ الفعلي لـ n عملية هو $\Theta(n)$.

نضيف الآن عملية المكدس MULTIPOP(S,k)، التي تُخرِج k غرضًا من قمة المكدس أو تفرغه كله إذا كان يحوي أغراضًا عددها أقل من k غرضًا. نفترض طبعًا أن k موجب، وإلا فإن عملية MULTIPOP لا تُحدِث أيَّ تغييرٍ في المكدس. في شبه الرماز التالي، تعيد العمليةُ STACK-EMPTY القيمةَ TRUE إذا لم يكن هناك أيُّ غرضٍ في المكدس عند استدعائها، و FALSE فيما عدا ذلك.

```
MULTIPOP(S, k)
1  while not STACK-EMPTY(S) and k > 0
2      POP(S)
3      k = k - 1
```

الشكل 1.17 أثر عمل MULTIPOP على مكدس S، مبيّن بدايةً في (أ). تنزع عمليةُ MULTIPOP$(S,4)$ الأغراضَ الأربعة في أعلى المكدس فيصبح المكدس كما في (**ب**). العملية التالية هي MULTIPOP$(S,7)$ وهي تفرغ المكدس – كما هو مبين في (**ت**) – لأن ما بقي فيه أقل من 7 أغراض.

يبيّن الشكل 1.17 مثالاً على MULTIPOP.

ما هو زمن تنفيذ MULTIPOP(S,k) على مكدس فيه s غرضًا؟ إن زمن التنفيذ الفعلي خطيٌّ بدلالة عدد عمليات POP المنفذة فعليًا، ولهذا السبب يكفي أن نحلّل MULTIPOP باستخدام الكلفة المجرّدة 1 لكل عملية PUSH وعملية POP. إن عدد التكرارات في حلقة **While** هو $\min(s,k)$ غرضًا منْزوعًا من المكدس. وفي كل تكرارٍ للحلقة يحدث استدعاءٌ واحد لعملية POP في السطر 2. إذن الكلفة الكلية لـ MULTIPOP هو $\min(s,k)$، وزمن التنفيذ الفعلي هو دالة خطيّة لهذه الكلفة.

دعونا ندرس متتالية من n عملية PUSH و POP وMULTIPOP على مكدس خالٍ بدايةً. إن كلفة عملية MULTIPOP في المتتالية هي $O(n)$ في أسوأ الحالات، وذلك لأن حجم المكدس هو على الأكثر n. لذا، فإن زمن أية عملية على المكدس هي في أسوأ الحالات $O(n)$، ونتيجةً لذلك، فإن كلفة n عملية هو $O(n^2)$، لأنه قد يكون لدينا $O(n)$ عملية MULTIPOP، كلفةُ كلٍّ منها $O(n)$. ومع أن هذا التحليل صحيح، فإن النتيجة التي يعطيها $O(n^2)$، بأن يأخذ كلفة أسوأ الحالات لكل عملية على حدة ليست محكمة كفاية.

يمكننا، باستخدام التحليل المجمَّع، الحصول على حدٍّ أعلى أفضل يأخذ بالاعتبار مجمل متتالية العمليات، وعددها n. والواقع أنه على الرغم من أن عملية MULTIPOP واحدة قد تكون مكلفة، فإن أية متتالية من n عملية PUSH و POP وMULTIPOP على مكدس خالٍ بدايةً تكلف على الأكثر $O(n)$. لماذا؟ لأن كلَّ غرضٍ يمكن أن يُنْزَع مرّةً واحدة على الأكثر لكل مرّة يُدفع به إلى المكدس. لهذا السبب فإن عدد المرات التي يمكن فيها استدعاء POP على مكدس غير خالٍ، ومن ضمنها الاستدعاءات داخل MULTIPOP، تساوي على الأكثر عدد عمليات PUSH، وهي على الأكثر n. أي إن أية متتالية من n عملية PUSH و POP و MULTIPOP تستغرق في المجمل زمنًا $O(n)$، مهما تكن قيمة n. ويكون متوسط كلفة عملية هو $O(n)/n = O(1)$. في التحليل المدمج، نسند لكلّ عمليةٍ كلفةً مخمّدةً هي الكلفة المتوسطة. إذن، في هذا المثال تكون الكلفة المخمّدة لكلٍّ من عمليات المكدس الثلاث هي $O(1)$.

نؤكد ثانيةً أنه على الرغم من أننا بينّا فقط أن الكلفة المتوسطة، ومن ثَمّ زمن التنفيذ، لعملية مكدس هي $O(1)$، إلا أننّا لم نستخدم أية محاكمة احتمالية. فقد عرضنا فعليًّا حدًّا في *أسوأ الحالات*، $O(n)$، على متتالية من n عملية. وبتقسيم هذه الكلفة الكلية على n، حصلنا على الكلفة المتوسطة للعملية الواحدة، أو ما يُعرَف بكلفتها المخمّدة.

زيادة عدّاد اثناني

لندرس مثالاً آخر للتحليل المجَمَّع، وهو مسألة تنجيز عداد اثناني من k بتًّا يَعُدُّ تصاعديًّا بدءًا من 0. نستخدم، لتمثيل العدّاد صفيفة بتات $A[0 \ldots k-1]$، حيث $A.length = k$. فإذا كان العدد الاثناني x مخزّنًا في العدّاد، فإن البت الأقل مرتبة منه يكون مخزّنًا في $A[0]$، فيما يُخزَّن البت الأعلى مرتبة في $A[k-1]$، بحيث يكون $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$. في البداية يكون $x = 0$، ومن ثَمّ فإن $A[i] = 0$ لكل القيم $i = 0, 1, \ldots, k-1$. ولإضافة 1 (بالمقاس 2^k) إلى القيمة في العدّاد، فإننا نستخدم الإجراء التالي.

```
INCREMENT(A)
1  i = 0
2  while i < A.length and A[i] == 1
3       A[i] = 0
4       i = i + 1
5  if i < A.length
6       A[i] = 1
```

يبيّن الشكل 2.17 ما يحدث لعدّاد اثناني عندما نَزيده 16 مرّة، بدءًا من القيمة البدائية 0، وانتهاءً بالقيمة 16. مع بداية كل تكرار في حلقة **while** في السطور 2-4، نريد أن نضيف 1 في الموقع i. فإذا كان $A[i] = 1$، فإن زيادة 1 تقلب البت في الموقع i إلى 0، وتعطي حَمْلاً قدره 1 لإضافته إلى الموقع $i + 1$ في التكرار التالي للحلقة. وإلاّ فإن الحلقة تتوقف، فإذا كان $i < k$، فإننا نعلم أن $A[i] = 0$، ولذلك فإن إضافة 1 في الموقع i يؤدي إلى قلب 0 إلى 1، وهذا ما يتكفّل به السطر 6. إن كلفةَ كلِّ عملية INCREMENT خطيّةٌ مع عدد البتات التي تتعرض للقلْب.

وكما في مثال المكدس، يعطينا تحليلٌ سريعٌ cursory حدًّا صحيحًا إلا أنه غير محكم كفاية. وإن تنفيذًا واحدًا لـ INCREMENT يستغرق زمنًا $\Theta(n)$ في أسوأ الحالات، وذلك عندما تحوي الصفيفة A القيمة 1 في كل المواقع. إذن، فإن متتالية من n عملية INCREMENT على عدّاد قيمته 0 بدايةً يستغرق زمنًا $O(nk)$ في أسوأ الحالات.

يمكننا أن نجعل تحليلنا أكثر دقةٍ ليعطي كلفةً في أسوأ الحالات $O(n)$ لمتتالية من n عملية INCREMENT، وذلك بملاحظة أنه لا تُقلَب كلُّ البتات عند كلِّ استدعاءٍ لـ INCREMENT. وكما يبيّن

قيمة العداد	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	الكلفة الكليّة
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

الشكل 2.17 عدّاد اثناني من 8 بتات تتزايد قيمته من 0 إلى 16 باستدعاء متتالية من 16 عملية INCREMENT. إن البتات التي تنقلب لأخذ القيمة التالية مظلّلة. وكلفة التنفيذ المقابلة لقَلْب البتات مبيّنة إلى يمين العدّاد. لاحظ أن الكلفة الكليّة لا تتجاوز أبدًا ضعف عدد عمليات INCREMENT.

الشكل 2.17، فإن $A[0]$ ينقلب عند كل استدعاء لـ INCREMENT. أما البت التالي له $A[1]$، فإنه ينقلب مرّة في كل استدعاءَيْن: أي إن متتاليةً من n عملية INCREMENT على عدّاد معدوم بدايةً تتسبب في قلب $A[1]$ $\lfloor n/2 \rfloor$ مرّة. وبالمثل، ينقلب البت $A[2]$ مرّةً كل أربع مرّات، أو $\lfloor n/4 \rfloor$ مرّة في متتالية من n عملية INCREMENT. وبوجه عام، عندما $i = 0, 1, \ldots, k-1$، ينقلب البت $A[i]$ فقط $\lfloor n/2^i \rfloor$ مرّة في متتالية من n عملية INCREMENT على عدّاد معدوم بدايةً. وإذا كان $i \geq k$، فإن البت $A[i]$ غير موجود أصلاً، وبالتالي لا يمكن له أن ينقلب. إذن، فالعدد الكلي للقلبات في المتتالية هو

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i}$$
$$= 2n ,$$

وذلك باستخدام المعادلة (أ.6). ونتيجةً لذلك، يكون زمن تنفيذ متتالية من n عملية INCREMENT على عدّاد معدوم بدايةً هو $O(n)$ في أسوأ الحالات. ومن ثَم تكون الكلفة المخمّدة للعملية الواحدة هي $O(n)/n = O(1)$.

تمارين

1-1.17

إذا تضمّنت مجموعة عمليات المكدس عملية MULTIPUSH تدفع k عنصرًا في المكدس، هل ستبقى الكلفة

المخمّدة لأية عملية مكدس محدّدة بالحدّ $O(1)$؟

2-1.17

بيّنْ أنه إذا تمّ تضمين عملية DECREMENT في مثال العداد ذي k بتًّا، فإن n عملية قد تكلّف ما قد يصل إلى زمن $\Theta(nk)$.

3-1.17

افترض أننا نُنفِّذ متتاليةً من n عملية على بنية معطيات ما. تكلّف العملية ذات الرقم i الكلفة i إذا كانت i إحدى قوى 2، وتكلّف 1 في باقي الحالات. استخدم التحليل المجمّع لتحديد الكلفة المخمّدة للعمليّة.

2.17 طريقة المحاسبة

نسند، في ***طريقة المحاسبة accounting method*** للتحليل المخمّد، كلفًا مختلفةً للعمليات المختلفة، مع تحميل بعض العمليات أكثر من كلفتها الفعلية أو أقل منها. تُسمّى الكلفة المحمّلة لعملية ما ***كلفتها المخمَّدة amortized cost***. عندما تتجاوز الكلفةُ المخمدة لعملية ما كلفتَها الفعلية، يُسنَد الفارق إلى أغراض معينة في بنية المعطيات ***كرصيد credit***. قد يساعد هذا الرصيد لاحقًا في التسديد لمصلحة عمليات تكون كلفتها المخمّدة أقل من كلفتها الفعليّة. إذن، يمكن أن نرى الكلفة المخمَّدة لعملية ما على أنها مفرّقة إلى كلفة فعليّة ورصيد إما أن يكون مودعًا أو مستهلكًا. قد يكون للعمليات المختلفة كلفًا مخمّدة متباينة، وبهذا تختلف هذه الطريقة عن التحليل المجمّع في أن لكلّ العملياتِ الكلفةَ المخمّدةَ نفسَها.

يجب أن نختار الكلف المخمّدة بعناية. فإذا كنا نريد تحليلاً بالكلف المخمّدة لنبيّن أن متوسط كلفة العملية الواحدة في أسوأ الحالات صغير، فعلينا أن نتحقق من أن الكلفة الكليّة المخمّدة لمتتالية من العمليات تمثّل حدًّا أعلى على الكلفة الكليّة الفعليّة لهذه المتتالية. إضافة إلى ذلك، وكما هو الحال في التحليل المجمَّع، يجب أن تكون هذه العلاقة محققةً على كل متتاليات العمليات. إذا أشرنا إلى الكلفة الفعلية للعملية i بـ c_i، وإلى كلفتها المخمدة بـ $\hat{c}_i$، فإننا نشترط أن تحقِّق كل المتتاليات المكونة من n عملية المتراجحة التالية:

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i \qquad (1.17)$$

إن الرصيد الكلي total credit المخزّن في بنية المعطيات هو الفارق بين الكلفة الكلية المخمّدة والكلفة الفعليّة، أي $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$. واعتمادًا على المتراجحة (1.17)، يجب أن يكون الرصيد الكلي الموافق لبنية المعطيات موجبًا على الدوام. وفي حال سُمِحَ للرصيد الكلي أن يصبح سالبًا (نتيجة تحميل أولى العمليات أقل من كلفتها مع الوعد بتسديد الحساب لاحقًا)، فستكون الكلف الكلية المخمدة المحققة في حينها أقل من الكلف الكلية الفعليّة المحققة؛ وعندها لن تكون الكلفة الكلية المخمدة فيما يخص متتالية العمليات حتى تلك

اللحظة، حدًا أعلى على الكلفة الكليّة الفعليّة. إذن، يجب أن نعير اهتمامنا كي لا يصبح الرصيد الكلي في بنية المعطيات سالبًا أبدًا.

عمليات المكدس

لإيضاح طريقة المحاسبة للتحليل المخمد، سنعود إلى مثال المكدس. تذكّر أن الكلفة الفعليّة للعمليات كانت:

PUSH 1 ,
POP 1 ,
MULTIPOP $\min(k, s)$,

حيث k هي المحدّد المعطى لـ MULTIPOP، و s حجم المكدس عند استدعائها. دعنا نسند الكلف المخمّدة التالية:

PUSH 2 ,
POP 0 ,
MULTIPOP 0 .

لاحظ أن الكلفة المخمّدة لـ MULTIPOP ثابتة (0)، فيما كلفتها الفعلية متغيّرة. في هذه الحالة، كل الكلف المخمّدة الثلاث ذات قيم ثابتة، مع أن الكلف المخمدة للعمليات المدروسة قد تتباين عمومًا بالمقاربة.

سنبيّن الآن أن بمقدورنا التسديد لأية متتالية من عمليات المكدس بتحميل الكلف المخمدة. افترض أننا نستخدم دولارًا لتمثيل كل وحدة من وحدات الكلفة. نبدأ بمكدسٍ فارغ. تذكّر التشابه الذي ذكرناه في المقطع 1.10 بين مكدس بنية المعطيات ومكدس الأطباق في كافتيريا. عندما نضيف طبقًا إلى المكدس، فإننا ندفع دولارًا لتسديد الكلفة الفعلية لهذه الإضافة، ويبقى معنا رصيد من دولار واحد (من الدولارين المدفوعين)، نضعه في الطبق. في أية لحظة، كل طبق في المكدس يحوي رصيدًا قدره دولار واحد.

إن الدولار المخزّن في الطبق هو تسديد مسبق لكلفة سحب الطبق من المكدس. وعندما ننفذ عملية POP، فإننا لا نحمّل العملية أية كلفة، ونسدّد كلفتها الفعلية باستخدام الرصيد المخزّن في المكدس. ولكي نسحب طبقًا، نأخذ دولار الرصيد من الطبق ونستخدمه للتسديد لعملية السحب. إذن بتحميل عملية PUSH أكثر قليلاً، لا نضطر إلى تحميل عملية POP أية كلفة.

إضافة إلى ذلك، يمكننا ألا نحمّل عمليات MUTIPOP أية كلفة. فلكي نسحب الطبق الأول، نأخذ دولار الرصيد من الطبق ونستخدمه لتسديد الكلفة الفعليّة لعملية POP، وهكذا دواليك.. وَلِنَسحْب الطبق الثاني، نأخذ ثانيةً دولار الرصيد من الطبق لتسديد الكلفة الفعليّة لعملية POP. وهكذا نكون قد حمّلنا على الدوام مقدَّمًا قدرًا كافيًا لتسديد عمليات MULTIPOP. وبعبارة أخرى، لما كان كل طبق من المكدس يحوي رصيدًا من دولار واحد، ولما كان المكدس يحوي عددًا موجبًا من الأطباق، كان *لأية* متتالية من n عملية MULTIPOP و POP و PUSH كلفة كلية مخمدة هي حدّ أعلى على الكلفة الكلية الفعلية. ولما كانت الكلفة

الكلية المخمدة من المرتبة $O(n)$، فهذه هي أيضًا حال الكلفة الكلية الفعلية.

زيادة عدّاد اثناني

ثمة إيضاحٌ آخرُ لطريقة المحاسبة يتمثّل في تحليل عمليّة INCREMENT على عدّاد اثناني يبدأ من الصفر. إن زمن تنفيذ هذه العملية، كما لاحظنا سابقًا، متناسب مع عدد البتات المقلوبة، وهذا ما سنستخدمه بوصفه الكلفة في هذا المثال. وسنستخدم الدولار مرّة أخرى لنمثّل واحدة الكلفة (قلب بت في هذا المثال).

للقيام بالتحليل المخمّد، نحمّل كلفة مخمّدة قدرها دولاران لجعل قيمة البت 1. عندما يأخذ بت القيمة 1 فإننا نستخدم دولارًا (من الدولارين المحصّلين) لتسديد الكلفة الفعلية لتغيير قيمة البت، ونضع الدولار الآخر على البت ليكون رصيدًا يُستخدَم لاحقًا عندما نقلب البت مجددًا إلى 0. في أية لحظة كل 1 في العدّاد لديه رصيد من دولار واحد عليه، وهكذا لا نحتاج إلى تحميله أية كلفة لإعادته إلى 0؛ فنحن نسدّد عملية العودة إلى الصفر باستخدام الدولار على البت.

يمكننا الآن تحديد الكلفة المخمّدة لـ INCREMENT. تُسدَّد عودة البتات إلى الصفر في حلقة **while** باستخدام الدولارات على البتات المصفَّرة. تغيّر INCREMENT على الأكثر بتًّا واحدًا إلى 1 في السطر 6، وهكذا تبلغ الكلفة المخمدة لعملية INCREMENT على الأكثر دولارين. إن عدد الوحدان في العدّاد لا يصبح سالبًا أبدًا، وهكذا فإن قيمة الرصيد تبقى دائمًا موجبة. إذن، الكلفة المخمدة الكليّة لـ INCREMENT هي $O(n)$، وهي التي تَحدّ الكلفة الفعليّة.

تمارين

1–2.17

افترض أننا نُنفِّذ متتالية من عمليات مكدس لا يتجاوز حجمه k أبدًا. بعد كل k عملية، نُعِدُّ نسخةً عن كامل المكدس لأغراض الخزن الاحتياطي. بيّنْ أن كلفة n عملية مكدس – ومن ضمنها نسْخ المكدس – هي $O(n)$، وذلك بإسناد الكلف المخمدة المناسبة لمختلف العمليات المنفذة على المكدس.

2–2.17

أعِدْ التمرين 3-1.17 باستخدام التحليل بطريقة المحاسبة.

3–2.17

افترض أننا لا نرغب في زيادة عداد ما فقط، بل بإعادته إلى الصفر أيضًا (أي بجعل كل بتاته تساوي 0). وبافتراض أن الزمن اللازم لقراءة بت أو تغييره هو $\Theta(1)$، بيّن كيف يمكن تنجيز عدادٍ كصفيفة من البتات بحيث تستغرق أية متتالية من n عملية INCREMENT و RESET زمنًا $O(n)$، وذلك على عداد معدوم بدايةً. (*تلميح*: احتفظ بمؤشر إلى البت 1 ذي المرتبة الأعلى.)

3.17 طريقة الكمون

بدلاً من تمثيل العمل المسدّد سلفًا كرصيد مخزّن في أغراض محدّدة في بنية المعطيات، تمثّل ***طريقة الكمون potential method*** للتحليل المخمّد العمل المسدّد سلفًا "كطاقة كامنة" أو فقط "كمون" يمكن تحريره لتسديد عمليات مستقبليّة. يُربط الكمون ببنية المعطيات ككل بدلاً من ربطه بأغراض محددة داخل البنية.

تعمل طريقة الكمون كالتالي. نُنجز n عملية، مبتدئين ببنية معطيات بدائية D_0. ولتكن c_i الكلفةَ الفعلية للعملية ذات الرقم i، لكل $i = 1, 2, \dots, n$. ولتكن D_i بنية المعطيات التي تنتج بعد تطبيق العملية ذات الرقم i على بنية المعطيات D_{i-1}. تقابل ***دالة كمون Potential function*** Φ كلَّ بنية معطيات D_i بعدد حقيقي $\Phi(D_i)$، هو الكمون المرافق لبنية المعطيات D_i. فتكون $\hat{c}_i$ ***الكلفة المخمدة amortized cost*** للعملية ذات الرقم i، بالاعتماد على دالة الكمون Φ معرّفة بـ

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \ . \qquad (2.17)$$

إذن، الكلفة المخمّدة لكل عمليّة هي كلفتها الفعليّة مضافًا إليها الزيادة في الكمون الناتج عن هذه العملية. واعتمادًا على المعادلة (2.17)، تكون الكلفة الكلية المخمدة للـ n عملية:

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \, . \qquad (3.17)\end{aligned}$$

تنتج المساواة الثانية من المعادلة (أ.9)، وذلك لأن الحدود $\Phi(D_i)$ تُحذف فيما بينها.

إذا استطعنا تعريف دالة كمون Φ بحيث يكون $\Phi(D_n) \ge \Phi(D_0)$، فإن الكلفة الكلية المخمدة $\sum_{i=1}^{n} \hat{c}_i$ تعطي حدًّا أعلى على الكلفة الكلية الفعلية $\sum_{i=1}^{n} c_i$. عمليًّا، لا نعرف دائمًا عدد العمليات التي يمكن القيام بها. ولذا إذا افترضنا $\Phi(D_i) \ge \Phi(D_0)$ لكل قيم i، فإننا نضمن، كما في طريقة المحاسبة، أن نسدّد سلفًا. نكتفي عادة بتعريف $\Phi(D_0)$ على أنه 0، ثمّ نبيّن أن $\Phi(D_i) \ge 0$ لكل قيم i. (انظر التمرين 17.3-1 للاطلاع على طريقة سهلة لمعالجة الحالات التي يكون فيها $\Phi(D_0) \ne 0$.)

نرى حدسيًّا أنه إذا كان فرق الكمون $\Phi(D_i) - \Phi(D_{i-1})$ للعملية i موجبًا، كانت الكلفة المخمدة $\hat{c}_i$ تمثل تحميلاً زائدًا للعملية i، وكان كمون بنية المعطيات في تزايد. وإذا كان فرق الكمون سالبًا، فإن الكلفة المخمدة تمثّل تحميلاً أقل من اللازم للعملية i، وتُسدَّد الكلفة الفعلية للعملية بإنقاص الكمون.

تعتمد الكلف المخمّدة المعرَّفة في المعادلتَين (2.17) و (3.17) على اختيار دالة الكمون Φ. إذ قد تُنتِج دوالُ كمون مختلفة كلفًا مخمدة مختلفة، إلا أنها كلها حدودٌ عليا للكلف الفعلية. في كثير من الأحيان هناك تسويات يمكن تحقيقها عند اختيار دالة الكمون؛ تعتمد أفضل دالة كمون يمكن استخدامها على حدود الزمن المرغوبة.

عمليات المكدس

لتوضيح طريقة الكمون، نعود مرة ثانية إلى مثال عمليات المكدس PUSH و POP و MULTIPOP. نعرِّف دالة الكمون Φ على مكدس على أنها عدد الأغراض في المكدس. إن دالة الكمون للمكدس الفارغ D_0 الذي نبدأ به هي $\Phi(D_0) = 0$. ولما كان من غير الممكن أن يكون عدد الأغراض في المكدس سالبًا، فإن للمكدس D_i الناتجة بعد العملية i كمونًا غير سالب، وهكذا فإن

$$\begin{aligned}\Phi(D_i) &\geq 0 \\ &= \Phi(D_0)\ .\end{aligned}$$

تمثل الكلفة الكلية المخمدة لـ n عملية بالاعتماد على Φ إذن حدًّا أعلى على الكلفة الفعلية.

لنحسب الآن الكلف المخمدة لمختلف عمليات المكدس. إذا كانت العملية ذات الرقم i على مكدس يحوي s غرضًا هي عملية PUSH، فإن فرق الكمون هو

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= (s+1) - s \\ &= 1\ .\end{aligned}$$

واعتمادًا على المعادلة (2.17) فإن الكلفة المخمدة لعملية PUSH هذه هي

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 1 \\ &= 2\ .\end{aligned}$$

لنفترض أن العملية ذات الرقم i على المكدس هي MULTIPOP(S, k)، وأن $k' = \min(k, s)$ هو عدد الأغراض المدفوعة خارج المكدس. إن الكلفة الفعلية للعملية هي k'، وفرق الكمون هو:

$$\Phi(D_i) - \Phi(D_{i-1}) = -k'\ .$$

إذن، الكلفة المخمدة لعملية MULTIPOP هي

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= k' - k' \\ &= 0\ .\end{aligned}$$

وبالمثل، فإن الكلفة المخمدة لعملية POP اعتيادية هي 0.

إن الكلفة المخمدة لكل من العمليات الثلاث هي $O(1)$، وهكذا فإن الكلفة الكلية المخمدة لمتتالية من n عملية هي $O(n)$. ولما كنا قد بينّا قبل قليل أن $\Phi(D_i) \geq \Phi(D_0)$، فإن الكلفة الكلية المخمدة لـ n عملية تمثل حدًّا أعلى للكلفة الكلية الفعلية. وبهذا تكون الكلفة في أسوأ الحالات لـ n عملية $O(n)$.

زيادة عدّاد الثنائي

سندرس مثالاً آخر لطريقة الكمون، وذلك بأن ننظر ثانيةً إلى عملية زيادة عداد اثنائي. نعرّف هذه المرة كمونَ

العدّاد b_i بعد عملية INCREMENT ذات الرقم i، بأنه عدد الخانات ذات القيمة 1 في العداد بعد العملية ذات الرقم i.

ولحساب الكلفة المخمدة لعملية INCREMENT، نفترض أن عملية INCREMENT ذات الرقم i تُصفِّر t_i بتًّا. فالكلفة الفعليّة للعملية هي إذن $t_i + 1$ على الأكثر، لأنه إضافة إلى تصفير t_i بتًّا، فإن العملية تضع 1 على الأكثر في بت واحد. فإذا كان $b_i = 0$، فهذا يعني أن العملية ذات الرقم i تُصفِّر كل البتات وعددها k، ويكون $b_{i-1} = t_i = k$. وإذا كان $b_i > 0$، فإن $b_i = b_{i-1} - t_i + 1$. وفي كلتا الحالتين، تتحقق المتراجحة $b_i \le b_{i-1} - t_i + 1$، ويكون فرق الكمون

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &\le (b_{i-1} - t_i + 1) - b_{i-1} \\ &= 1 - t_i \ .\end{aligned}$$

وعلى هذا تكون الكلفة المخمّدة

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\le (t_i + 1) + (1 - t_i) \\ &= 2 \ .\end{aligned}$$

إذا بدأ العدّاد بالقيمة صفر، فإن $\Phi(D_0) = 0$. ولما كان $\Phi(D_i) \ge 0$ مهما كانت i، فإن الكلفة الكلية المخمدة لمتتالية من n عملية INCREMENT هي $O(n)$.

تعطينا طريقة الكمون أسلوبًا بسيطًا لتحليل العداد، ولو كان لا يبدأ بالصفر. يبدأ العداد بـ b_0 بتًّا قيمتها 1، وبعد n عملية INCREMENT، يصبح b_n بتًّا قيمتها 1 حيث $0 \le b_0$ و $b_n \le k$. (تذكَّر أن k هو عدد البتات في العدّاد) يمكننا إعادة كتابة المعادلة (3.17) كالتالي

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \Phi(D_n) + \Phi(D_0) \ . \tag{4.17}$$

لدينا $\hat{c}_i \le 2$ مهما كانت $1 \le i \le n$. ولما كان $\Phi(D_0) = b_0$ و $\Phi(D_n) = b_n$، فإن الكلفة الفعليّة لـ n عملية INCREMENT هي

$$\begin{aligned}\sum_{i=1}^{n} c_i &\le \sum_{i=1}^{n} 2 - b_n + b_0 \\ &= 2n - b_n + b_0 \ .\end{aligned}$$

نلاحظ، بوجه خاص، أن الكلفة الفعليّة الكليّة هي $O(n)$، مادام $k = O(n)$، وذلك لأن $b_0 \le k$. وبعبارة أخرى، إذا نفّذنا $n = \Omega(k)$ عملية INCREMENT على الأقل، فإن الكلفة الكليّة الفعليّة هي $O(n)$، مهما كانت قيمة العدّاد البدائية.

تمارين

1-3.17

افترض أن لدينا دالة كمون Φ بحيث يكون $\Phi(D_i) \geq \Phi(D_0)$ مهما كانت i، إلا أن $\Phi(D_0) \neq 0$. بيّن أنه توجد دالة كمون Φ' بحيث يكون $\Phi'(D_0) = 0$ و $\Phi'(D_i) \geq 0$، مهما كانت $i \geq 1$، وأن الكلف المخمّدة باستخدام Φ' هي نفسها الكلف المخمدة باستخدام Φ.

2-3.17

أعدْ التمرين 3-1.17 باستخدام التحليل بطريقة الكمون.

3-3.17

ليكنْ لدينا بنية معطيات كومة-أصغر ثنائية عاديّة مع n عنصرًا تدعم عمليتَي INSERT و EXTRACT-MIN بزمن $O(\lg n)$ في أسوأ الحالات. أعطِ دالة كمون Φ حيث تكون الكلفة المخمدة لعملية INSERT $O(\lg n)$، والكلفة المخمدة لعملية EXTRACT-MIN هي $O(1)$، وبيّنْ أن ذلك يعمل كما ينبغي.

4-3.17

ما هي الكلفة الكليّة لتنفيذ n عملية مكدس MULTIPOP و POP و PUSH بافتراض أن المكدس يبدأ وفيه s_0 غرضًا وينتهي وفيه s_n غرضًا؟

5-3.17

افترض أن عدّادًا يبدأ بعددٍ من b بتًّا قيمتها 1 بتمثيله الاثناني، عوضًا عن أن يبدأ من 0. بيّن أن كلفة تنفيذ n عملية INCREMENT هو $O(n)$ إذا كان $n = \Omega(b)$. (لا تفترض أن b عدد ثابت.)

6-3.17

بيّنْ كيف يمكن تنجيز رتل مع عمليتَي مكدس معتادتين (التمرين 6-1.10) بحيث تكون الكلفة المخمدة لكل عملية ENQEUE وكل عملية DEQUEUE هي $O(1)$.

7-3.17

صمّمْ بنية معطيات تدعم العمليتين التاليتين على مجموعة ديناميكية S من الأعداد الصحيحة:

$\text{INSERT}(S, x)$ التي تُدْخِل العنصرَ x في S.

$\text{DELETE-LARGER-HALF}(S)$ التي تحذف من S أكبر $\lceil |S|/2 \rceil$ عنصرًا.

اشرح كيف يمكن تنجيز بنية المعطيات هذه بحيث تُنفَّذ أية متتالية من m عملية بزمن $O(m)$. ينبغي أن يتضمن تنجيزك أيضًا طريقةً لإخراج العناصر S في زمن $O(|S|)$.

4.17 الجداول الديناميكيّة

في بعض التطبيقات، لا نعرف سلفًا عدد الأغراض التي ستُخزَّن في جدولٍ ما؛ فقد نخصص حجمًا لجدول ما، لنكتشف فيما بعد أنه غير كافٍ. في هذه الحالة، يجب إعادة تخصيص حجمٍ أكبر للجدول، ونقل كل الأغراض المخزّنة في الجدول الأول ونسخها في الجدول الجديد الأوسع. وبالمثل، إذا حُذِفت أغراض كثيرة من الجدول، فقد يكون من المجدي إعادة تخصيص حجم أصغر للجدول. ندرس في هذا المقطع هذه المسألة التي تُعنى بتوسيع جدولٍ ما وتقليصه ديناميكيًّا. ونبيّن باستخدام التحليل المخمّد أن الكلفة المخمّدة للإدراج والحذف هي فقط $O(1)$، ولو كانت الكلفة الفعليّة لعملية ما كبيرة عندما تطلق توسيعًا للجدول أو تقليصًا له. إضافة إلى ذلك، سنرى كيف يمكن ضمان ألا يتجاوز الحيّز غير المستخدم في الجدول الديناميكي، أبدًا، جزءًا ثابتًا من حجمه الكلي.

نفترض أن الجدول الديناميكي يدعم عمليتَي TABLE-INSERT و TABLE-DELETE. تُدرِج عملية TABLE-INSERT عنصرًا داخل الجدول ليشغلَ ***شقبًا slot*** واحدًا، وهو الحيّز المحدّد لعنصر واحد. وبالمثل، تُعنى العملية TABLE-DELETE بحذف عنصر من الجدول، وتحرّر بذلك شقبًا. لا أهمية لتفاصيل بنية المعطيات المستخدمة لتنظيم هذا الجدول؛ فقد نستخدم مكدسًا (مقطع 1.10)، أو كومة (الفصل 6) أو جدول تلبيد (الفصل 11). وقد نستخدم أيضًا صفيفة أو مجموعة من الصفيفات لتنجيز خزن الأغراض كما فعلنا في المقطع 3.10.

سنجد من المناسب استخدام مفهوم عَرَفناه عندما حلّلنا التلبيد (الفصل 11). نعرّف ***عامل التحميل load factor*** $\alpha(T)$ لجدول غير خالٍ T على أنه عدد العناصر المخزنة في الجدول مقسّمة على حجمه (عدد الشقوب فيه). نسند إلى الجدول الخالي (الذي لا يحوي أية عناصر) الحجم 0، ونعرّف عامل تحميله على أنه 1. إذا كان عامل التحميل لجدول ديناميكي محدودًا تحت ثابتٍ ما، فإن الحجم غير المستخدم في الجدول لن يتجاوز أبدًا جزءًا ثابتًا من الحجم الكلي.

سنبدأ بتحليل جدولٍ ديناميكيٍ لا ننجز عليه إلا عمليات إدراج. ثم ندرس حالة أكثر عمومية يُسمَح فيها بالقيام بعمليات إدراج وحذف.

1.4.17 توسيع الجدول

لنفترض أن الجدول يخزَّن في صفيفة من الشقوب. ويمتلئ الجدول عندما تكون كل الشقوب قد استُخدِمت، أو بصورةٍ مكافئة، عندما يكون عامل تحميله مساويًا 1.[1] في بعض البيئات البرمجية، إذا جَرَتْ محاولة لإدراج

[1] قد نرغب في بعض الحالات، عندما يتعلق الأمر مثلاً بجدول تلبيد مفتوح العناوين، أن نعتبر أن جدولاً ما ملآن عندما يساوي عامل تحميله ثابتًا أصغر من 1 تمامًا. (انظر التمرين 17.4-1)

عنصر في جدول ملآن، فالبديل الوحيد هو استبعاد المحاولة برسالة خطأ. سنفترض، على أية حال، أن البيئة البرمجية التي نتعامل معها، مثل العديد من البيئات الحديثة، تتيح نظامًا لإدارة الذاكرة قادرًا على تخصيص كتل خزن وتحريرها عند الطلب. وهكذا عندما يُدرَج عنصرٌ في جدولٍ ملآن، فإن بمقدورنا ***توسيع expand*** الجدول بتخصيص جدولٍ جديدٍ فيه شقوب أكثر عددًا من الجدول القديم. ولما كنا دائمًا بحاجة إلى أن يكون الجدول مخزنًا في ذاكرة متتالية، كان علينا تخصيص صفيفة جديدة للجدول الأكبر ثم نسخ العناصر من الجدول القديم إلى الجدول الجديد.

هناك كسبيّة شائعة common heuristic تتمثل في تخصيص جدولٍ جديدٍ فيه ضعف شقوب الجدول القديم. فإذا لم يكن هناك إلا عمليات إدراج، فإن عامل التحميل للجدول 1/2 على الأقل دائمًا، وبهذا فإن مقدار حجم الذاكرة المهدور لا يتجاوز أبدًا نصف حجم الجدول.

نفترض، في شبه الرماز التالي، أن T غرضٌ يمثّل الجدول. يحتوي الواصفة $T.table$ مؤشرًا إلى كتلة الخزن التي تمثل الجدول، والحقل $T.num$ عدد العناصر في الجدول، والحقل $T.size$ عدد الشقوب الكلية في الجدول. في البداية يكون الجدول فارغًا: $T.num = T.size = 0$.

```
TABLE-INSERT(T, x)
 1  if T.size == 0
 2      allocate T.table with 1 slot
 3      T.size = 1
 4  if T.num == T.size
 5      allocate new-table with 2 · T.size slots
 6      insert all items in T.table into new-table
 7      free T.table
 8      T.table = new-table
 9      T.size = 2 · T.size
10  insert x into T.table
11  T.num = T.num + 1
```

لاحظ أن لدينا إجراءَي "إدراج" هنا: إجراء TABLE-INSERT نفسه ***والإدراج الأساسي elementary insertion*** في جدول، في السطرين 6 و 10. يمكن أن نحلّل زمن تنفيذ TABLE-INSERT بدلالة عدد مرات الإدراج الأساسيّة بإسناد كلفة قيمتها 1 إلى كل عملية إدراج أساسية. نفترض أن زمن التنفيذ الفعلي لـ TABLE-INSERT خطيٌّ بدلالة زمن إدراج العناصر الإفرادية، وبهذا تكون الكلفة المضافة اللازمة لتحصيص الجدول الابتدائي في السطر 2 ثابتة، بحيث تفوق كلفةُ نقل العناصر في السطر 6 الكلفةَ المضافة اللازمة لتحصيص منطقة التخزين وتحريرها في السطرين 5 و 7. نسمّي الحدث الذي تنفَّذ عنده السطور 5-9 ***توسيعًا expansion***.

لنحلّل الآن متتالية من n عملية TABLE-INSERT مطبقة على جدولٍ خالٍ بدايةً. ما هي كلفة العملية c_i ذات الرقم i؟ إذا كان ما يزال هناك مكان في الجدول الحالي (أو إذا كانت هذه هي العملية الأولى)، فإن ، إذ إننا بحاجة إلى تنفيذ عملية إدراج أساسية واحدة فقط في السطر 10. أما إذا كان الجدول الحالي ملآنًا، وحدث توسيع، فإن $c_i = i$: كلفة تساوي 1 للإدراج الأساسي في السطر 10 يُضاف إليها $i - 1$ لنسخ العناصر من الجدول القديم إلى الجدول الجديد في السطر 6. إذا نُفِّذت n عملية، فإن كلفة عملية في أسوأ الأحوال هي $O(n)$، وهذا ما يؤدي إلى حدٍّ أعلى من المرتبة $O(n^2)$ لزمن التنفيذ الكلي لـ n عملية TABLE-INSERT.

إن هذا الحد ليس محكمًا كفاية، لأننا قلّما نوسّع الجدول خلال n عملية إدراج. تتسبب العملية i تحديدًا في إحداث توسيعٍ فقط عندما يكون $i - 1$ من قوى 2 الصحيحة. إن الكلفة المخمّدة لعملية ما هي في الحقيقة $O(1)$، كما سنرى باستخدام التحليل المجمَّع. إن كلفة العملية ذات الرقم i هي

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2 ,} \\ 1 & \text{otherwise .} \end{cases}$$

إذن، الكلفة الكليّة لـ n عملية TABLE-INSERT هي

$$\begin{aligned} \sum_{i=1}^{n} c_i &\le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \\ &< n + 2n \\ &= 3n , \end{aligned}$$

وذلك، لأن هناك n عملية على الأكثر كلفةُ كلٍّ منها 1، وكلف باقي العمليات تشكل سلسلة هندسية. ولما كانت الكلفة الكلية لـ n عملية TABLE-INSERT هي $3n$، كانت الكلفة المخمّدة للعملية الواحدة هي على الأكثر 3.

يمكننا، باستخدام طريقة المحاسبة، أن نستشعر بوضوح أكبر لماذا يجب أن تكون الكلفة المخمدة لعملية TABLE-INSERT 3. إن كل عنصر يسدِّد كلفة 3 عمليات إدراج أساسيّة: إدراج العنصر نفسه في الجدول الحالي، ونقل نفسه عند توسيع الجدول، وتحريك عنصر آخر كان قد انتقل مرّة من قبل عندما توسّع الجدول. افترض على سبيل المثال، أن حجم الجدول هو m بعد عملية التوسيع مباشرة. إذن عدد العناصر في الجدول هو $m/2$، ولا يمتلك الجدول أي رصيد. نحمّل كل عملية إدراج 3 دولارات. يكلّف الإدراج الأساسي الذي يحدث فورًا دولارًا واحدًا. ونضع دولارًا آخر في رصيد العنصر المضاف. ونضيف الدولار الثالث إلى رصيد أحد العناصر التي عددها $m/2$ الموجودة في الجدول. لن يمتلئ الجدول ثانية حتى نضيف $m/2 - 1$ عنصرًا إضافيًا، وهكذا عندما يحوي الجدول m عنصرًا ويصبح ملآنًا، نكون قد وضعنا في رصيد كل عنصر دولارًا يدفعه ليعيد إدراج نفسه أثناء عملية التوسيع.

يمكن أيضًا استخدام طريقة الكمون لتحليل متتالية من n عملية TABLE-INSERT، ويجب أن نستخدمها في المقطع 17.4-2 لنصمّم عملية TABLE-DELETE كلفتُها المخمدة هي أيضًا $O(1)$. نبدأ بتعريف دالة كمون Φ تكون معدومةً بعد عملية التوسيع مباشرة، إلا أنها تزداد حتى تصبح قيمتها مساوية لطول الجدول في اللحظة التي يصبح فيها الجدول ملآنًا، وبهذا يمكننا التسديد لعملية التوسيع التالية باستخدام هذا الكمون. إن الدالة

$$\Phi(T) = 2 \cdot T.num - T.size \tag{17.5}$$

هي أحد الخيارات. ويكون لدينا $T.num = T.size/2$، بعد عملية التوسيع مباشرة. وهكذا يكون $\Phi(T) = 0$ مثلما نرغب. أما قبل عملية التوسيع مباشرة، فيكون $T.num = T.size$، وهكذا يكون $\Phi(T) = T.num$، مثلما نرغب. إن القيمة البدائية للكمون هي 0، ولما كان الجدول على الدوام نصف ملآن على الأقل، فإن $T.num \geq T.size/2$ وهذا يقتضي أن تكون الدالة $\Phi(T)$ موجبة دومًا. وهكذا فإن مجموع الكلف المخمدة لـ n عملية TABLE-INSERT يمثّل حدًّا أعلى لمجموع الكلف الفعلية.

لتحليل الكلفة المخمدة لعملية TABLE-INSERT ذات الرقم i، نجعل num_i يمثّل عددَ العناصر المخزنة في الجدول بعد العملية ذات الرقم i، و $size_i$ يمثّل الحجمَ الكلي للجدول بعد العملية ذات الرقم i، و Φ_i الكمون بعد العملية ذات الرقم i. بدايةً، يكون لدينا $num_0 = 0$ و $size_0 = 0$ و $\Phi_0 = 0$.

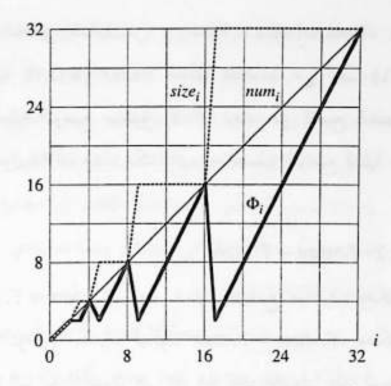
إذا لم تتسبب العملية ذات الرقم i في توسيعٍ ما، يكون لدينا $size_i = size_{i-1}$، وتكون الكلفة المخمدة لهذه العملية:

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2(num_i - 1) - size_i) \\
&= 3 \,.
\end{aligned}$$

أما إذا تسببت العملية ذات الرقم i في توسيعٍ ما، فيكون عندها $size_i = 2 \cdot size_{i-1}$ و $size_{i-1} = num_{i-1} = num_i - 1$، وهذا يقتضي أن $size_i = 2 \cdot (num_i - 1)$. إذن الكلفة المخمدة للعملية هي:

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - (2(num_i - 1) - (num_i - 1)) \\
&= num_i + 2 - (num_i - 1) \\
&= 3 \,.
\end{aligned}$$

يبيّن الشكل 17.3 قيمة num_i، و $size_i$، و Φ_i بدلالة i. لاحظ كيف أن الكمون يتزايد ليسدّد توسيع الجدول.



الشكل 3.17 أثر متتالية من n عملية TABLE-INSERT على عدد العناصر num_i في الجدول، وعلى عدد الشقوب في الجدول $size_i$، وعلى الكمون $\Phi_i = 2 \cdot num_i - size_i$، حيث تقاس كل هذه المقادير بعد العملية ذات الرقم i. يبيّن الخط الرفيع num_i، والخط المتقطع $size_i$، والخط الثخين Φ_i. لاحظ أنه قبل عملية التوسيع مباشرة، يزداد الكمون حتى تبلغ قيمتُه عدد العناصر في الجدول، وبهذا يمكنه أن يسدِّد لنقْل كلّ العناصر إلى الجدول الجديد. بعد ذلك، يهبط الكمون إلى 0، إلا أنه يرتفع مباشرة إلى 2 عند إدراج العنصر الذي تسبّب في التوسيع.

2.4.17 توسيع الجدول وتقليصه

إن حذف عناصر محدّدة من الجدول بهدف تنجيز عملية TABLE-DELETE بسيط جدًّا. غير أننا كثيرًا ما نرغب في ***تقليص contract*** الجدول عندما يصبح عامل تحميل الجدول صغيرًا جدًّا، وذلك للحدّ من حجم الذاكرة المهدور. يماثل تقليص الجدول توسيعه: عندما ينخفض عدد العناصر كثيرًا، فإننا نخصص جدولاً جديدًا أصغر منه، ثم ننسخ العناصر من الجدول القديم في الجدول الجديد. ويمكن عندها تحرير حيّز التخزين للجدول القديم بإعادته إلى نظام إدارة الذاكرة. نريد في الوضع المثالي أن نحافظ على الخاصيتين التاليتين:

- عامل تحميل الجدول محدود من الأسفل بثابت.
- والكلفة المخمدة لعمليات الجدول محدودة من الأعلى بثابت.

نفترض أننا نقيس الكلفة بدلالة عمليات إدراج وحذف أساسيّة.

قد تعتقد أن علينا مضاعفة حجم الجدول عند إضافة عنصر إلى جدولٍ ملآن، وتقليص حجمه إلى النصف عندما تتسبب عملية حذف في جعل الجدول أقل من نصف ملآن. تضمن هذه الاستراتيجيّة ألاَّ ينخفض عامل التحميل أبدًا إلى أقل من 1/2، ولكنها لسوء الحظ قد تتسبب في جعل الكلفة المخمدة لعملية ما كبيرًا فعلاً. لنأخذ السيناريو التالي: نقوم بـ n عملية على جدولٍ T، حيث n هي قوة صحيحة لـ 2. إن نِصْف العمليات الأولى هي عمليات إدراج، وهي تكلّف وفق تحليلنا السابق كلفةً كلية $\Theta(n)$. في نهاية متتالية

الإدراج، يكون $T.num = T.size = n/2$. وفيما يخص النصف الثاني من العمليات، فإننا ننفذ المتتالية التالية:

insert, delete, delete, insert, insert, delete, delete, insert, insert, ...

تتسبب عملية الإدراج الأولى في توسيع الجدول إلى الحجم n. وتتسبب عمليتا الحذف التاليتان في تقليص الجدول من جديد إلى الحجم $n/2$. تتسبب عمليتا إدراج جديدتان في توسيع آخر، وهكذا. كلفة كل عملية توسيع وتقليص هي $\Theta(n)$، وهناك $\Theta(n)$ عملية منهما. وهكذا فإن الكلفة الكلية لـ n عملية هي $\Theta(n^2)$، ومن ثَم، فإن الكلفة المخمدة لعملية واحدة هي $\Theta(n)$.

إن سيئة هذه الاستراتيجيّة واضحة: بعد عملية توسيع، لا نقوم بعددٍ كافٍ من عمليات الحذف لنسدّد للتقليص. وبالمثل، بعد التقليص لا نقوم بعددٍ كافٍ من عمليات الإدراج لنسدّد للتوسيع.

يمكننا تحسين هذه الاستراتيجيّة بالسماح لعامل التحميل بالهبوط إلى أقل من 1/2. وعلى وجه الخصوص، تتابع مضاعفة حجم الجدول عند إدراج عنصر في جدول ملآن، إلا أننا نقسم حجم الجدول على 2 عندما تتسبب عملية حذف في جعل الجدول ملآنًا إلى أقل من ربعه، بدلاً من نصفه. وبهذا يكون عامل التحميل محدودًا من الأسفل بالثابت 1/4.

قد نعتبر بالحدس أن عامل تحميل يساوي 1/2 مثاليًا، ويكون عندها كمون الجدول 0. ومع ابتعاد عامل التحميل عن 1/2 يتزايد الكمون، وبذلك عندما يحين أوان توسيع الجدول أو تقليصه، يكون الجدول قد اكتسب ما يكفي من الكمون ليسدّد لنسخ كل العناصر في الجدول الجديد المحصص. إذن، نحتاج إلى دالة كمون تصل إلى $T.num$ عندما يكون عامل التحميل قد ازداد إلى 1 أو تناقص إلى 1/4. وبعد توسيع الجدول أو تقليصه، يعود عامل التحميل مجددًا إلى 1/2، ويهبط الكمون عائدًا إلى 0.

لن ندرج هنا رماز TABLE-DELETE، إذ إنه مشابه لرماز TABLE-INSERT. وسنفترض لأغراض التحليل، أنه عندما ينخفض عدد العناصر في الجدول إلى 0، فإننا نحرّر حيّز خزن الجدول، أي إذا كان $T.num = 0$ فإن $T.size = 0$.

بإمكاننا الآن استخدام طريقة الكمون لتحليل كلفة متتالية من n عملية TABLE-INSERT و TABLE-DELETE. نبدأ بتعريف دالة كمون Φ تصبح 0 بعد التوسيع أو التقليص مباشرة، وتزداد بازدياد عامل التحميل إلى 1 أو تَناقصه إلى 1/4. نسمّي عامل التحميل لجدول T غير خال بـ $\alpha(T) = T.num/T.size$. ولما كان $T.num = T.size = 0$ و $\alpha(T) = 1$ للجدول الخالي، فلدينا دومًا $T.num = \alpha(T) \cdot T.size$، سواءٌ أكان الجدول خاليًا أم لا. سنستخدم دالة الكمون

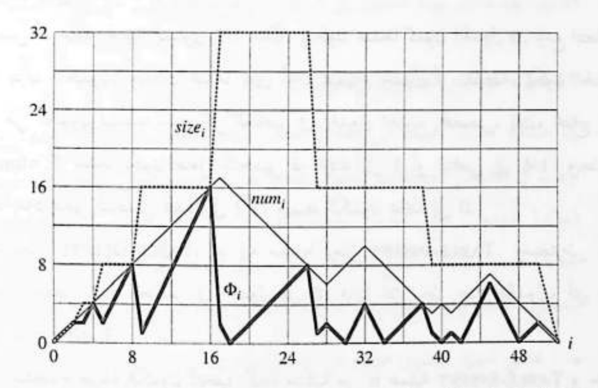
$$\Phi = \begin{cases} 2 \cdot T.num - T.size & \text{if } \alpha(T) \geq 1/2 \, , \\ T.size/2 - T.num & \text{if } \alpha(T) < 1/2 \, . \end{cases} \tag{6.17}$$

لاحظ أن كمون الجدول الخالي هو 0، وأن الكمون لا يصبح سالبًا أبدًا. إذن، فالكلفة المخمدة الكليّة لمتتالية

من عمليات اعتمادًا على Φ هي حدٌّ أعلى للكلفة الفعليّة لهذه المتتالية.

سنتوقف قليلاً قبل البدء بالقيام بتحليل دقيق، لنراقب بعض خواص دالة الكمون المبينة في الشكل 4.17. لاحظ أنه عندما يكون عامل التحميل 1/2، يكون الكمون 0. وعندما يكون عامل التحميل 1، يكون لدينا $T.size = T.num$، وهذا يقتضي أن $\Phi(T) = T.num$، وبذلك يمكن للكمون أن يسدّد للتوسيع فيما إذا أُدرج عنصرٌ ما. عندما يكون عامل التحميل 1/4، يكون لدينا $T.size = 4 \cdot T.num$، وهذا يقتضي أن $\Phi(T) = T.num$، وبذلك يمكن للكمون أن يسدِّد للتقليص فيما إذا حُذِفَ عنصرٌ ما.

لتحليل متتالية من n عملية TABLE-INSERT و TABLE-DELETE، نفترض أن c_i الكلفة الفعلية للعملية ذات الرقم i، و $\hat{c}_i$ الكلفة المخمّدة اعتمادًا على Φ، و num_i عدد العناصر المخزنة في الجدول بعد العملية ذات الرقم i، و $size_i$ حجم الجدول الكلي بعد العملية ذات الرقم i، و α_i عامل التحميل للجدول بعد العملية ذات الرقم i، و Φ_i الكمون بعد العملية ذات الرقم i. بدايةً، يكون $num_0 = 0$، و $size_0 = 0$، و $\alpha_0 = 1$، و $\Phi_0 = 0$.



الشكل 4.17 أثر متتالية من n عملية TABLE-INSERT و TABLE-DELETE على num_i عدد العناصر في الجدول، و $size_i$ $size_i$ عدد الشقوب في الجدول، والكمون

$$\Phi_i = \begin{cases} 2 \cdot num_i - size_i & \text{if } \alpha_i \geq 1/2 , \\ size_i /2 - num_i & \text{if } \alpha_i < 1/2 . \end{cases}$$

حيث يقاس كل من هذه القيم بعد العملية ذات الرقم i. يبيّن الخط الرفيع num_i، والخط المتقطع $size_i$، والخط الثخين Φ_i، لاحظ أنه قبل توسيع الجدول مباشرةً، يكون الكمون قد تنامى حتى أصبح يساوي عدد العناصر في الجدول، وبهذا يمكنه التسديد لنقل كل العناصر إلى الجدول الجديد. وبالمثل، قبل التقليص مباشرة، يكون الكمون قد تنامى حتى أصبح يساوي عدد العناصر في الجدول.

نبدأ بالحالة التي تكون فيها العملية ذات الرقم i هي TABLE-INSERT. إن التحليل الذي قمنا به مماثل للتحليل الذي عالجنا فيه توسيع الجدول في المقطع 1-4.17 عندما $\alpha_{i-1} \geq 1/2$. وسواءٌ أتوسّع الجدول أم لا، فإن كلفة العملية المخمدة $\hat{c}_i$ هي 3 على الأكثر. وإذا كان $\alpha_{i-1} < 1/2$، فلن يكون للجدول أن يتوسع بنتيجة لهذه العملية، إذ لا يمكن له التوسّع إلا عندما $\alpha_{i-1} = 1$. وكذلك، إذا كان $\alpha_i < 1/2$، فإن الكلفة المخمدة للعملية ذات الرقم i هي:

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\
&= 1 + (size_i/2 - num_i) - (size_i/2 - (num_i - 1)) \\
&= 0\ .
\end{aligned}$$

إذا كان $\alpha_{i-1} < 1/2$، ولكن $\alpha_i \geq 1/2$ ، فإن

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (size_{i-1}/2 - num_{i-1}) \\
&= 1 + (2(num_{i-1} + 1) - size_{i-1}) - (size_{i-1}/2 - num_{i-1}) \\
&= 3 \cdot num_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&= 3\alpha_{i-1} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&< \frac{3}{2} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&= 3\ .
\end{aligned}$$

وهكذا، فإن الكلفة المخمدة لعملية TABLE-INSERT هي 3 على الأكثر.

نلتفت الآن إلى الحالة التي تكون فيها العملية ذات الرقم i هي TABLE-DELETE. في هذه الحالة، يكون $num_i = num_{i-1} - 1$. إذا كان $\alpha_{i-1} < 1/2$، وجب أن ندرس تَسَبُّب العملية في تقليصٍ ما. فإذا لم تكن هذه هي الحالة، فإن $size_i = size_{i-1}$، وتكون الكلفة المخمدة للعملية:

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\
&= 1 + (size_i/2 - num_i) - (size_i/2 - (num_i + 1)) \\
&= 2\ .
\end{aligned}$$

وإذا كان $\alpha_{i-1} < 1/2$ وتَسبَّبت العملية ذات الرقم i في تقليص الجدول، فإن الكلفة الفعلية للعملية هي $c_i = num_i + 1$، لأننا نحذف عنصرًا واحدًا وننقل num_i عنصرًا. ويكون لدينا $size_i/2 = size_{i-1}/4 = num_{i-1} = num_i + 1$، والكلفة المخمدة للعملية هي:

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= (num_i + 1) + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\
&= (num_i + 1) + \big((num_i + 1) - num_i\big) - ((2 \cdot num_i + 2) - (num_i + 1)) \\
&= 1 \ .
\end{aligned}$$

وعندما تكون العملية ذات الرقم i هي TABLE-DELETE، و $\alpha_{i-1} \geq 1/2$، فإن الكلفة المخمدة محدودة أيضًا من الأعلى بثابت، والتحليل متروك للتمرين 17.4-2.

والخلاصة هي أنه، لما كانت الكلفة المخمدة لكل عملية محدودة من الأعلى بثابت، فإن الزمن الفعلي لأية متتالية من n عملية على جدول ديناميكي هي $O(n)$.

تمارين

17.4-1

افترض أننا نريد تنجيز جدول تلبيد ديناميكي مفتوح العناوين. لماذا ينبغي أن نعتبر أن الجدول ملآن عندما يصل عامل تحميله إلى قيمةٍ ما α أصغر تمامًا من 1؟ صِفْ بإيجاز كيف تجعل الإدراج في جدول تلبيد ديناميكي مفتوح العناوين يُنفَّذ بطريقة تجعل توقع الكلفة المخمدة لعملية الإدراج هي $O(1)$. لماذا لا يكون توقع الكلفة الفعلية لعملية الإدراج بالضرورة $O(1)$ لكل عمليات الإدراج؟

17.4-2

بيّنْ أنه إذا كان $\alpha_{i-1} \geq 1/2$ وكانت العملية ذات الرقم i على الجدول الديناميكي هي TABLE-DELETE، فإن الكلفة المخمدة للعملية اعتمادًا على دالة الكمون في (17.6) محدودة من الأعلى بثابت.

17.4-3

افترض أننا بدلاً من أن نقلِّص جدولاً ما بتقسيم حجمه على 2 عندما يهبط عامل تحميله إلى أقل من 1/4، نقلِّصُه بضرب حجمه بـ 2/3 عندما يهبط عامل تحميله إلى 1/3. وذلك باستخدام دالة الكمون

$$\Phi(T) = |2 \cdot T.num - T.size| \ ,$$

بيّنْ أن الكلفة المخمدة لعملية TABLE-DELETE التي تعتمد هذه الاستراتيجية محدودة من الأعلى بثابت.

مسائل

17-1 عداد اثناني معكوس البتات

يدرس الفصل 30 خوارزميةً هامةً اسمها تحويل فورييه السريع، أو FFT. تقوم أول خطوة من الخوارزمية FFT بتنفيذ ***تبديل يعكس البتات bit-reversal permutation*** على جدول دخل $A[0\,..\,n-1]$، حيث

$n = 2^k$، حيث k عدد صحيح غير سالب. يقوم هذا التبديل بمبادلة كل عنصرين من الجدول تمثيلُ دليل أحدهما الاثناني عكس تمثيل دليل الثاني الاثناني.

يمكننا أن نعبّر عن كل دليل a بمتتالية من k بتًّا $\langle a_{k-1}, a_{k-2}, \ldots, a_0 \rangle$، حيث $a = \sum_{i=0}^{k-1} a_i 2^i$. نعرّف $\text{rev}_k(\langle a_{k-1}, a_{k-2}, \ldots, a_0 \rangle) = \langle a_0, a_1, \ldots, a_{k-1} \rangle$;

إذن

$$\text{rev}_k(a) = \sum_{i=0}^{k-1} a_{k-i-1} 2^i .$$

فمثلاً، إذا كان $n = 16$ (أو مُكافئه $k = 4$)، فإن $\text{rev}_k(3) = 12$، وذلك لأن التمثيل ذي الأربع بتات لـ 3 هو 0011، والذي يصبح عند عكسه 1100، وهو التمثيل ذي الأربع بتات لـ 12.

أ. لتكن rev_k دالةً تُنفَّذ في زمن $\Theta(k)$، اكتب خوارزميةً لتنفيذ التبديل الذي يعكس البتات لصفيفة طولها $n = 2^k$ في زمن $O(nk)$.

يمكننا استخدام خوارزمية تعتمد على التحليل المخمد لنحسّن زمن تنفيذ التبديل الذي يعكس البتات. نحافظ على "عدّاد معكوس البتات" وعلى إجراء BIT-REVERSED-INCREMENT الذي يعطي عند إعطائه قيمة a للعدّاد المعكوس البتات، القيمة $\text{rev}_k(\text{rev}_k(a) + 1)$. فإذا كان، مثلاً $k = 4$ والعدّاد المعكوس البتات يبدأ عند الصفر، فإن الاستدعاءات المتتالية لـ BIT-REVERSED-INCREMENT تولدّ المتتالية:

$0000, 1000, 0100, 1100, 0010, 1010, \ldots = 0, 8, 4, 12, 2, 10, \ldots$.

ب. افترض أن الكلمات في حاسوبك تخزّن قيمًا من k بتًّا، وأن بمقدور حاسوبك معالجة القيم الاثنانية في واحدة الزمن بتطبيق عمليات مثل الانزياح يسارًا أو يمينًا بمقادير اعتباطية، والعطف على مستوى البت bitwise-AND، والفصل على مستوى البت bitwise-OR، إلخ. صِفْ تنجيزًا لإجراء BIT-REVERSED-INCREMENT يسمح بتنفيذ تبديل يعكس البتات على جدول من n عنصرًا بزمن كليّ $O(n)$.

ت. افترض أنه يمكنك إزاحة كلمة يسارًا أو يمينًا بتًّا واحدًا فقط في واحدة الزمن. هل مازال ممكنًا تنجيز تبديل يعكس البتات في زمن $O(n)$.

2-17 جعل البحث الثنائي ديناميكيًا

يستغرق البحث الثنائي لصفيفة مفروزة زمنًا لغاريتميًّا للقيام بالبحث، إلا أن زمنَ إدراج عنصرٍ جديدٍ خطيٌّ بدلالة حجم الجدول. يمكننا تحسين زمن الإدراج بالاحتفاظ بعدّة جداول مفروزة.

لنفترض، بوجه خاص، أننا نرغب في دعم SEARCH و INSERT على مجموعة من n عنصرًا. وليكن

$k = \lceil \lg(n+1) \rceil$، وليكن التمثيل الاثناني لـ n $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$. لدينا k صفيفة مفروزة $A_0, A_1, \ldots, A_{k-1}$، حيث طول الصفيفة A_i هو 2^i لكل $i = 0, 1, \ldots, k-1$. إن كل صفيفة ستكون ملأى أو فارغة، تبعًا لكون $n_i = 1$ أو $n_i = 0$ على الترتيب. وبهذا يكون عدد العناصر المخزنة في الصفيفات، التي عددها k، هو $\sum_{i=0}^{k-1} n_i 2^i = n$. ومع أن كلَّ صفيفةٍ مستقلةٍ مفروزةٌ بحدِّ ذاتها، فلا توجد علاقة محددة بين عناصر صفيفات مختلفة.

أ. صِفْ كيف يمكن أداء عملية SEARCH في بنية المعطيات هذه. وحلّل زمن تنفيذها في أسوأ الحالات.

ب. صِفْ كيف يمكن إدراج عنصر جديد في بنية المعطيات هذه. وحلّل زمن تنفيذ هذه العملية في أسوأ الحالات، وزمن تنفيذها المخمَّد.

ت. ناقش كيف يمكن تنجيز DELETE.

3-17 الأشجار ذات الثقل المتوازن المخمدة

لنأخذ شجرة بحث ثنائية عادية، ونُغْنِها بأن نضيف إلى كل عقدة x الواصفة $x.size$ الذي يعطي عدد المفاتيح المخزنة في الشجرة الفرعية التي جذرها x. ليكن α ثابتًا في المجال $1/2 \le \alpha < 1$. نقول عن عقدة معطاة x إنها ***α−متوازنة*** ***α-balanced*** إذا كان $x.left.size \le \alpha \cdot x.size$ و $x.right.size \le \alpha \cdot x.size$. ونقول عن شجرة بأسرها أنها ***α−متوازنة*** إذا كانت كل عقدة من عقدها α−متوازنة. اقترح G. Varghese النهج المخمَّد التالي للمحافظة على أشجار متوازنة الثقل.

أ. إن شجرة 1/2-متوازنة ، بمعنى ما، هي شجرة متوازنة قدر الإمكان. إذا كان لديك عقدة x في شجرة بحث ثنائية اعتباطية، بيِّنْ كيف يُمكن إعادة بناء الشجرة الفرعية التي جذرها x بحيث تصبح 1/2-متوازنة. يجب أن تُنفَّذ خوارزميتك في زمن $\Theta(x.size)$ وبمقدورها استخدام $O(x.size)$ مساحة خزن إضافية مساعِدة.

ب. بيِّنْ أن إجراء بحث في شجرة بحث ثنائية α−متوازنة من n عقدة، يستغرق في أسوأ الحالات زمنًا $O(\lg n)$.

افترض فيما تبقّى من هذه المسألة أن الثابت α أكبر تمامًا من 1/2. لنفترض أننا أنجزنا الإجراءَين INSERT و DELETE بالطريقة المعتادة للتعامل مع شجرة بحث ثنائية من n عقدة، إلا أنهما وبعد كل عملية، إذا فقدت إحدى العقد في الشجرة خاصية α−متوازنة، فإنه يُعاد بناء الشجرة الفرعية التي جذرها أعلى عقدة غير متوازنة، كهذه العقدة، لتصبح 1/2-متوازنة.

سوف نحلِّل نهج إعادة البناء هذا باستخدام طريقة الكمون. لتكن x عقدةً في شجرة بحث ثنائية T،

نعرّف

$$\Delta(x) = |x.left.size - x.right.size| \ ,$$

ونعرّف كمون T على أنه

$$\Phi(T) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta(x) \ ,$$

حيث c ثابت كبير كفاية ويتعلّق بـ α.

ت. ناقش أن كمون أية شجرة بحث ثنائية غير سالب، وأن كمون الشجرة 1/2-متوازنة معدوم.

ث. افترض أن m وحدةً كمونٍ كافيةٌ لتسديد إعادة بناء شجرة فرعية ذات m عقدة. كم يجب أن يكون كِبر c بدلالة α، حتى يستغرق إعادة بناء شجرة فرعية غير α–متوازنة زمنًا مخمدًا $O(1)$؟

ج. بيّن أن كلفة إدراج عقدة في شجرة ذات n عقدة α–متوازنة أو حذفها منها هو $O(\lg n)$.

4-17 *كلفة إعادة تنظيم بنية أشجار حمراء–سوداء*

هناك أربع عمليات أساسية على الأشجار الحمراء–سوداء تُجري ***تغييرات بنيوية structural modifications***: إدراج العقد، وحذف العقد، والدورانات وتغيير اللون. رأينا سابقًا أن RB-INSERT و RB-DELETE يَستخدمان $O(1)$ دورانًا فقط، وإدراج عقدٍ وحذف عقدٍ للمحافظة على الخواص الحمراء–السوداء، إلا أنهما قد يقومان بالمزيد من عمليات تغيير اللون.

أ. صِفْ شجرةً حمراء–سوداء نظاميّة ذات n عقدة بحيث يتسبّب استدعاء RB-INSERT في إدراج العقدة ذات الرقم $n+1$ فيها عمليات تغيير لون من الرتبة $\Omega(\lg n)$. ثم صِفْ شجرةً حمراء–سوداء نظاميّة ذات n عقدة بحيث يتسبّب استدعاء RB-DELETE في حذف عقدة محددة عمليات تغيير لون من الرتبة $\Omega(\lg n)$.

على الرغم من أن عدد عمليات تغيير اللون قد يكون لغاريتميًّا للعملية الواحدة في أسوأ الحالات، إلا أننا سنبرهن أن أية متتالية من m عملية RB-INSERT و RB-DELETE على شجرة حمراء–سوداء خاليةٍ بدايةً تتسبب في $O(m)$ تغييرًا بنيويًّا في أسوأ الحالات. لاحظ أننا نعدّ كل تغيير لون تغييرًا بنيويًّا.

ب. إن بعض الحالات التي تعالجها الحلقة الرئيسة في رماز كلٍّ من RB-INSERT-FIXUP و RB-DELETE-FIXUP هي حالات ***إنهاء تنفيذ terminating*** أي عندما تتحقق مثل هذه الحالة يتوقف تنفيذ الحلقة بعد عدد ثابتٍ من العمليات الإضافية. حدِّد أيَّ الحالات في كل من RB-INSERT-FIXUP و RB-DELETE-FIXUP هي حالات إنهاء تنفيذ، وأيّها ليس حالات إنهاء. (*تلميح:* انظر إلى الأشكال 5.13 و 6.13 و 7.13.)

سندرس أولاً التغييرات البنيوية عند القيام بعمليات إدراج فقط. لتكن T شجرة حمراء–سوداء، ونعرّف $\Phi(T)$ على أنه عدد العقد الحمراء فيها. افترض أنه يمكن لوحدة كمون واحدة أن تسدّد للتغيرات البنيوية التي تحدث في أية حالة من حالات RE-INSERT-FIXUP الثلاث.

ت. لتكن T' الشجرة الناتجة من تطبيق الحالة 1 من RB-INSERT على T. ناقش أن $\Phi(T') = \Phi(T) - 1$.

ث. عندما ندرج عقدة في شجرة حمراء–سوداء باستخدام RB-INSERT، يمكننا تجزيء العملية إلى ثلاثة أجزاء. اسرد التغييرات البنيوية والتغيّرات الكمونية الناتجة عن الأسطر 1-16 من RB-INSERT، للحالات التي ليست حالات إنهاء في RB-INSERT-FIXUP، وأخيرًا لحالات الإنهاء في RB-INSERT-FIXUP.

ج. ناقش، اعتمادًا على الجزء (ت) أن عدد التغييرات البنيوية المخمّدة المنفذة عند أي استدعاء لـ RB-INSERT هي $O(1)$.

نرغب الآن في أن نبرهن أن هناك $O(m)$ تغييرًا بنيويًا عندما توجد عمليات إدراج وحذف. نعرّف لكل عقدة x،

$$w(x) = \begin{cases} 0 & \text{إذا كانت } x \text{ حمراء} \\ 1 & \text{إذا كانت } x \text{ سوداء وليس لها أبناء حمر} \\ 0 & \text{إذا كانت } x \text{ سوداء ولها ابن واحد أحمر} \\ 2 & \text{إذا كانت } x \text{ سوداء ولها ابنان أحمران} \end{cases}$$

نعيد الآن تعريف كمون شجرة حمراء–سوداء T على أنه

$$\Phi(T) = \sum_{x \in T} w(x) \ ,$$

ولتكن T' الشجرة الناتجة عن تطبيق أية حالة ماعدا حالات الإنهاء في RB-INSERT-FIXUP أو RB-DELETE-FIXUP على T.

ح. بيّن أن $\Phi(T') \le \Phi(T) - 1$ في كل الحالات التي لا يحدث فيها إنهاء في RB-INSERT-FIXUP. ناقش أن عدد التغييرات البنيوية المخمّدة المنفّذة عند أي استدعاء لـ RB-INSERT-FIXUP هو $O(1)$.

خ. بيّن أن $\Phi(T') \le \Phi(T) - 1$ في كل الحالات التي لا يحدث فيها إنهاء في RB-DELETE-FIXUP. ناقش أن عدد التغييرات البنيوية المخمّدة المنفّذة عند أي استدعاء لـ RB-DELETE-FIXUP هو $O(1)$.

د. أكمل برهان أن أية متتالية من m عملية RB-INSERT و RB-DELETE تُنْجِز، في أسوأ الحالات، $O(m)$ تغييرًا بنيويًا.

5-17 التحليل التنافسي للقوائم الذاتية التنظيم باستخدام كسبية النقل–إلى–المقدمة

القائمة الذاتية التنظيم ***self-organizing list*** هي قائمة مترابطة من n عنصرًا، ولكلٍّ من هذه العناصر مفتاحٌ وحيد. عندما نبحث عن عنصر في القائمة، نُعطَى مفتاحًا، ونحاول العثور على العنصر الذي له هذا المفتاح.

للقائمة الذاتية التنظيم خاصيتان هامتان:

1. للعثور على عنصر في القائمة اعتمادًا على مفتاحه المعطى، علينا عبور القائمة من البداية وحتى نصادف العنصر ذا المفتاح المعطى. فإذا كان هذا العنصر في المرتبة k من بداية القائمة، فإن كلفة العثور عليه هي k.
2. قد نعيد ترتيب عناصر القائمة بعد أية عملية، وفقًا لقاعدة معطاة بكلفة محددة. وقد نختار أية كسبية نريد لنقرر كيف نعيد ترتيب القائمة.

افترض أننا نبدأ بقائمة معطاة من n عنصرًا، وأننا أُعطينا متتالية نفاذ مؤلفة من المفاتيح $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_m \rangle$ للبحث عنها بالترتيب. كلفة المتتالية هي مجموع كلف النفاذ إلى كل مفتاح منها على حدة.

تركِّز هذه المسألة، من بين كل الطرق المختلفة الممكنة لإعادة ترتيب القائمة بعد عملية ما، على المبادلة بين عناصر القائمة المتجاورة – أي تبديل مواقعها في القائمة – بكلفة قدرها واحد لكل عملية مبادلة. نبيِّن فيما يلي – باستخدام دالة كمون – أن الكلفة الكلية لكسبية محددة لإعادة ترتيب القائمة، وهي كسبية النقل–إلى–المقدمة، لا تتجاوز أربع أضعاف أية كسبية أخرى يمكن استخدامها للمحافظة على ترتيب القائمة — ولو كانت الكسبية الأخرى على معرفة مسبقة بمتتالية النفاذ! نسمِّي هذا النوع من التحليل ***بالتحليل التنافسي*** ***competitive analysis***.

نرمز إلى كلفة النفاذ إلى متتالية σ مع كسبية محددة H وترتيب بدئي للقائمة، بالرمز $C_H(\sigma)$. ليكن m عدد مرات النفاذ في σ.

أ. ناقش أنه إذا لم تكن الكسبية H تعرف متتالية النفاذ مسبقًا، فإن كلفة الحالة الأسوأ لـ H على متتالية نفاذ σ هي $C_H(\sigma) = \Omega(mn)$.

عند استخدام كسبية ***النقل–إلى–المقدمة*** ***move-to-front***، بعد البحث عن عنصر x مباشرة، نقوم بنقل x إلى الموقع الأول في القائمة (أي مقدمة القائمة).

لنرمز بـ $\text{rank}_L(x)$ إلى مرتبة العنصر x في القائمة L، أيْ موقع x في القائمة L. فإذا كان x، على سبيل المثال، العنصرَ الرابع في L، فإن $\text{rank}_L(x) = 4$. ولنرمز بـ c_i إلى كلفة النفاذ إلى σ_i باستخدام كسبية

النقل-إلى-المقدمة، التي تتضمن كلفة إيجاد العنصر في القائمة وكلفة نقله إلى مقدمة القائمة باستخدام سلسلة من المبادلات بين عناصر القائمة المتجاورة.

ب. بيِّن أنه إذا كان σ_i يسمح بالنفاذ إلى العنصر x في القائمة L باستخدام كسبية النقل-إلى-المقدمة، فإن $c_i = 2 \cdot \text{rank}_L(x) - 1$.

نقارن الآن النقل-إلى-المقدمة بأية كسبية أخرى H تعالج متتالية نفاذ وفقًا للخاصتين المذكورتين في البداية. قد تقوم الكسبية H بمبادلة العناصر في القائمة بالطريقة التي ترتئيها، وقد تكون على معرفة مسبقة بكامل متتالية النفاذ.

لتكن L_i القائمة بعد النفاذ إلى σ_i باستخدام النقل-إلى-المقدمة، ولتكن L_i^* القائمة بعد النفاذ إلى σ_i باستخدام الكسبية H. نرمز إلى كلفة النفاذ إلى σ_i باستخدام النقل-إلى-المقدمة بـ c_i، وباستخدام الكسبية H، بـ c_i^*. افترض أن الكسبية H تقوم بـ t_i^* مبادلة عند النفاذ إلى σ_i.

ت. لقد بيّنتَ في الجزء (ب) أن $c_i = 2 \cdot \text{rank}_{L_{i-1}}(x) - 1$. بيّن الآن أن $c_i^* = 2 \cdot \text{rank}_{L_{i-1}^*}(x) + t_i^*$.

نعرّف ***الزوج المعكوس inversion*** في القائمة L_i على أنه زوج من العناصر y و z حيث يسبق العنصر y العنصر z في القائمة L_i، فيما يسبق العنصر z العنصر y في القائمة L_i^*. افترض أن القائمة L_i فيها q_i زوجًا معكوسًا بعد معالجة متتالية النفاذ $\langle \sigma_1, \sigma_2, \dots, \sigma_i \rangle$. في هذه الحالة نعرّف دالة كمون Φ تقابل كل قائمة L_i بالعدد الحقيقي $\Phi(L_i) = 2q_i$. على سبيل المثال، إذا كانت القائمة L_i تضم العناصر $\langle e, c, a, d, b \rangle$، وكانت القائمة L_i^* تضم العناصر $\langle c, a, b, d, e \rangle$، فإن القائمة L_i تضم خمسة أزواج معكوسة $((e, c), (e, a), (e, d), (e, b), (d, b))$، وبهذا يكون $\Phi(L_i) = 10$. لاحظ أن $\Phi(L_i) \geq 0$ مهما كانت i، وأن كلاًّ من النقل-إلى-المقدمة والكسبية H تبدأان مع القائمة نفسها L_0، إذن $\Phi(L_0) = 0$.

ث. ناقش أن أية مبادلة إما أن تزيد الكمون بمقدار 2 أو تُنقصه بـ 2.

افترض أن النفاذ σ_i يعثر على العنصر x. ولكي نعرف كيف يتغير الكمون بسبب σ_i، سنجزّئ العناصر ما عدا x إلى 4 مجموعات، اعتمادًا على مواقعها في القوائم تمامًا قبل النفاذ ذي الترتيب i:

- المجموعة A، تتألف من العناصر التي تسبق x في كلٍّ من L_{i-1} و L_{i-1}^*.
- المجموعة B، تتألف من العناصر التي تسبق x في L_{i-1} وتتبعه في L_{i-1}^*.
- المجموعة C، تتألف من العناصر التي تتبع x في L_{i-1} وتسبقه في L_{i-1}^*.
- المجموعة D، تتألف من العناصر التي تتبع x في كلٍّ من L_{i-1} و L_{i-1}^*.

ج. ناقش أن $\text{rank}_{L_{i-1}}(x) = |A| + |B| + 1$ وأن $\text{rank}_{L^*_{i-1}}(x) = |A| + |C| + 1$.

ح. بيّن أن النفاذ σ_i يغيّر الكمون بالمقدار

$$\Phi(L_i) - \Phi(L_{i-1}) \leq 2(|A| - |B| + t_i^*)$$

حيث، كما ذكرنا سابقًا، تقوم الكسبية H بـ t_i^* مبادلة خلال النفاذ σ_i.

عرّف كلفة النفاذ σ_i المخمّدة $\hat{c}_i$ بـ $\hat{c}_i = c_i + \Phi(L_i) - \Phi(L_{i-1})$.

خ. بيّن أن كلفة النفاذ σ_i المخمّدة $\hat{c}_i$ محدودة من الأعلى بالمقدار $4c_i^*$.

د. استنتج أن $C_{\text{MTF}}(\sigma)$ كلفة النفاذ إلى مفاتيح المتتالية σ باستخدام النقل-إلى-المقدمة هي على الأكثر أربعة أضعاف $C_H(\sigma)$ كلفة النفاذ إلى مفاتيح المتتالية σ باستخدام أية كسبية H، بفرض أن كلتا الكسبيتين تبدأان مع القائمة نفسها.

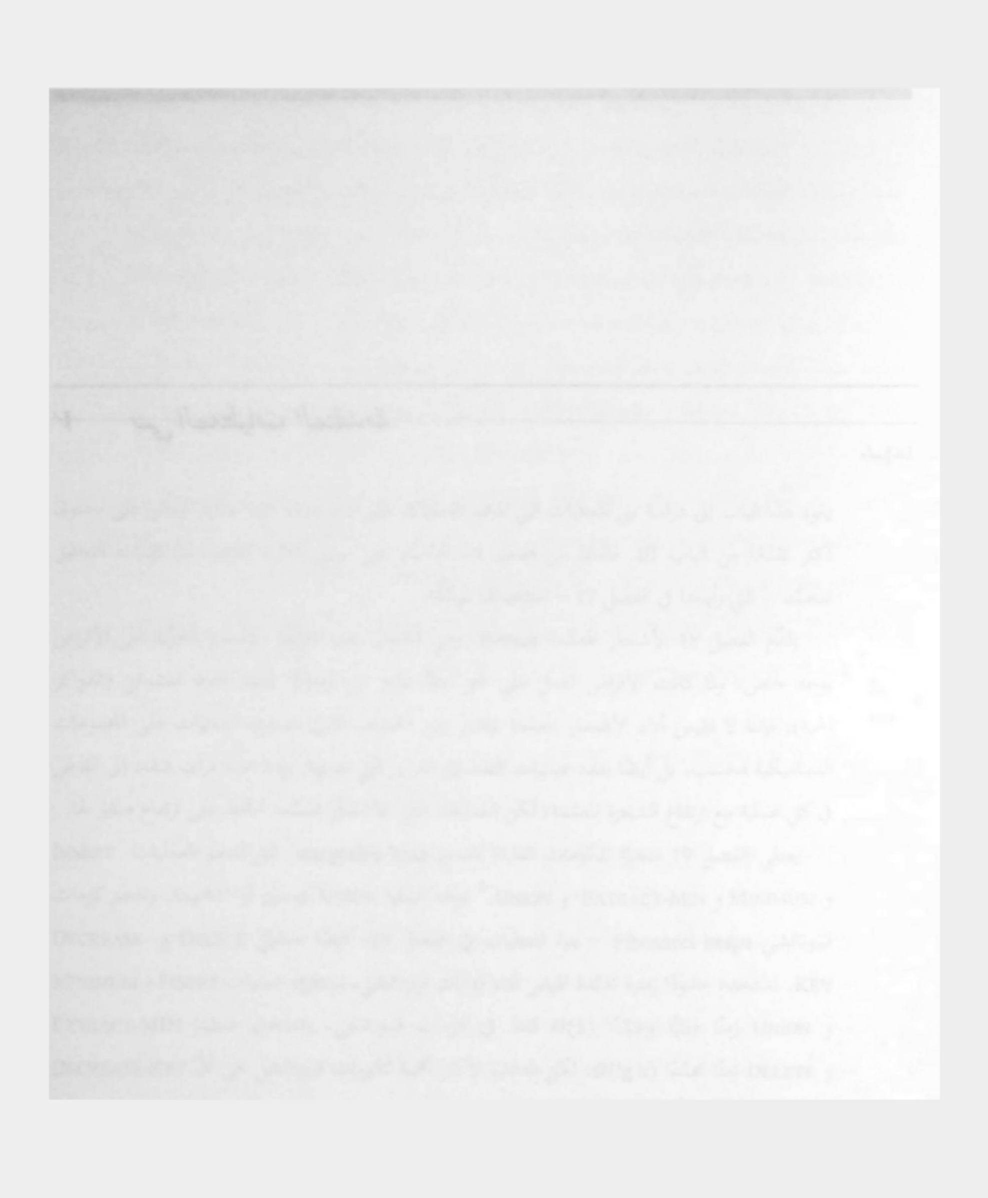
ملاحظات الفصل

استخدم Aho و Hopcroft و Ullman [5] التحليل المجمّع لتحديد زمن تنفيذ عمليات على غابة من المجموعات المنفصلة disjoint-set؛ سنحلل بنية المعطيات هذه باستخدام طريقة الكمون في الفصل 21. يقوم Tarjan [331] بمسح لطريقتي المحاسبة والكمون في التحليل المخمد، ويقدّم تطبيقات عدّة. وهو ينسب طريقة المحاسبة إلى عدّة مؤلفين، منهم M.R. Brown و R.E. Tarjan و S. Huddleston و K. Mehlhorn. وينسب طريقة الكمون إلى D.D. Sleator. ويعود مصطلح "مخمد amortized" إلى D.D. Sleator و R.E. Tarjan.

إن دوال الكمون ذات فائدةٍ أيضًا في برهان حدود دنيا في بعض أنماط المسائل. نعرّف لكل تشكيلة من المسألة، دالة كمون تقابل بين التشكيلة وعدد حقيقي. ثم نحدّد كمون التشكيلة البدائية Φ_{init}، وكمون التشكيلة النهائية Φ_{final}، والتغيّر الأعظم في الكمون $\Delta\Phi_{\max}$ الناتج عن أية خطوة، وعليه لا بد أن يكون عدد الخطوات على الأقل $|\Phi_{\text{final}} - \Phi_{\text{init}}|/\Delta\Phi_{\max}$. هناك أمثلة عن استخدام دوال الكمون في برهان حدود دنيا لتعقيد عمليات I/O في أعمال Cormen و Sundquist و Wisniewski [80]، و Floyd [107] و Aggarwal و Vitter [4]. طبّق Krumme و Cybenko و Venkataraman [221]، دوال الكمون لبرهان حدود دنيا على ***البث الشامل gossiping***: تبادل عنصر وحيد من كل عقدة في بيان إلى باقي العقد الأخرى.

تعمل كسبية النقل-إلى-المقدمة من المسألة 17-5 جيدًا في الحالة العملية. يضاف إلى ذلك أننا إذا أقررنا

بأننا عندما نعثر على عنصر، فإننا نستخرجه من موقعه في القائمة ونضعه في مقدمتها في زمن ثابت، فبإمكاننا أن نبرهن أن كلفة النقل-إلى-المقدمة هي على الأكثر ضعفي كلفة أية كسبية أخرى، ومن ضمنها، مرة أخرى، الكسبية التي تَعْرف مسبقًا كامل متتالية النفاذ.



V بنى المعطيات المتقدمة

تمهيد

يعود هذا الباب إلى دراسة بنى المعطيات التي تدعم العمليات على المجموعات الديناميكية، ولكن على مستوى أكثر تقدمًا من الباب III. فاثنان من فصول هذا الباب، على سبيل المثال، يَستخدمان تقنيات التحليل المخمَّد – التي رأيناها في الفصل 17 – استخدامًا موسَّعًا.

يقدِّم الفصل 18 الأشجار المعمَّمة B-trees، وهي أشجار بحث متوازنة صُمِّمت لتُخزَّن على الأقراص بوجه خاص. ولما كانت الأقراص تعمل على نحو أبطأ بكثير من الذواكر ذات النفاذ العشوائي (الذواكر الحية)، فإننا لا نقيس أداء الأشجار المعمَّمة بمقدار زمن الحساب الذي تستغرقه العمليات على المجموعات الديناميكية فحسب، بل أيضًا بعدد عمليات النفاذ إلى القرص التي تؤديها. يزداد عدد مرات النفاذ إلى القرص في كل عملية مع ارتفاع الشجرة المعمَّمة، لكن العمليات على الأشجار المعمَّمة تحافظ على ارتفاع صغير لها.

يعطي الفصل 19 تنجيزًا للكومات القابلة للدمج mergeable heap، التي تدعم العمليات INSERT و MINIMUM و EXTRACT-MIN و UNION.[1] توحّد عملية UNION كومتَيْن أو تدمجهما. وتدعم كومات فيبوناتشي Fibonacci heaps – بنية المعطيات في الفصل 19– أيضًا عمليتَيّ DELETE و -DECREASE KEY. نستخدم حدودًا زمنية مخمَّدة لقياس أداء كومات فيبوناتشي. تستغرق عمليات INSERT و MINIMUM و UNION زمنًا فعليًا ومخمَّدًا $O(1)$ فقط في كومات فيبوناتشي، وتستغرق عمليتا EXTRACT-MIN و DELETE زمنًا مخمَّدًا $O(\lg n)$. لكن الفائدة الأكثر أهميةً لكومات فيبوناتشي هي أنَّ DECREASE-KEY

[1] كما في المسألة 10-2، عرَّفنا كومة قابلة للدمج لتدعم MINIMUM و EXTRACT-MIN، وبذلك يمكننا أيضاً تسميتها *كومات قابلة للدمج وفق الأصغر mergeable min-heap*. بالمقابل، إذا كانت تدعم MAXIMUM و EXTRACT-MAX، فستكون *كومات قابلة للدمج وفق الأكبر mergeable min-heap*. نقصد بالكوماتِ القابلةِ للدمجِ الكوماتِ القابلةَ للدمج وفق الأصغر، ما لَم نُشر إلى خلاف ذلك.

تستغرق زمنًا مخمَّدًا $O(1)$ فقط. ولما كانت عملية DECREASE-KEY تستغرق زمنًا مخمَّدًا ثابتًا، فإنَّ كومات فيبوناتشي هي مركَّبات أساسية في بعض أسرع الخوارزميات مقاربةً حاليًّا في مسائل البيانات.

مع ملاحظة أننا يمكن أن نتغلب على الحد الأدنى للفرز، $\Omega(n \lg n)$، حين تكون المفاتيح أعدادًا صحيحة ضمن مجال محدد، يتساءل الفصل 20 عن إمكان تصميم بنية معطيات تدعم عمليات المجموعات الديناميكية SEARCH و INSERT و DELETE و MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR بزمنٍ $o(\lg n)$ عندما تكون المفاتيح أعدادًا صحيحة ضمن مجال محدد. وتشير الإجابة إلى أننا نستطيع ذلك، باستخدام بنية معطيات عودية معروفة باسم شجرة van Emde Boas. إذا كانت المفاتيح أعدادًا صحيحة فريدة مسحوبة من المجموعة $\{0, 1, 2, \ldots, u-1\}$ حيث u قوة تامة للعدد 2، فإنَّ أشجار van Emde Boas تدعم العمليات المذكورة بزمنٍ $O(\lg \lg u)$.

أخيرًا، يقدِّم الفصل 21 بنية للمجموعات المنفصلة. لدينا فضاء من n عنصرًا مجزَّأة إلى مجموعات ديناميكية. في البداية، ينتمي كل عنصر إلى مجموعته الوحيدة العنصر. توحّد عملية UNION مجموعتين، ويحدِّد الاستعلام FIND-SET المجموعة الوحيدة التي تتضمن عنصرًا معينًا حاليًّا. بتمثيل كل مجموعة بشجرة بسيطة ذات جذر نحصل على عمليات مدهشة السرعة: سلسلة من m عمليةً تُنفَّذ بزمن $O(m\,\alpha(n))$، حيث $\alpha(n)$ هي دالة بطيئة النمو جدًّا - $\alpha(n)$ هي على الأكثر 4 في أي تطبيق يمكن تخيله. التحليل المخمَّد الذي يثبت هذا الحد الزمني معقَّد جدًّا بمقدار بساطة بنية المعطيات.

ليست المواضيع المقدَّمة في هذا الباب هي الأمثلة الوحيدة على بنى المعطيات "المتقدمة" في أي حال من الأحوال. فثمة بنى أخرى للمعطيات المتقدمة تتضمن ما يلي:

- ***الأشجار الديناميكية Dynamic trees***، أدخلها Sleator و Tarjan [319] وناقشها Tarjan [330]، وهي تحتفظ بغابة من الأشجار المنفصلة ذات الجذر. كلُّ وصلة في كلِّ شجرة لها تكلفة ذات قيمة حقيقية. تدعم الأشجار الديناميكية استعلامات العثور على الآباء، والجذور، وتكاليف الوصلات، والوصلة الأقل تكلفة في طريق بسيط من عقدة ما إلى الجذر. يمكن معالجة الأشجار بقطع الوصلات، وتحديث تكلفة جميع الوصلات ضمن طريق بسيط من عقدةٍ ما إلى الجذر، ووصل جذر إلى شجرة أخرى، وتحويل عقدة ما إلى جذر في الشجرة التي تظهر فيها. تعطي إحدى تنجيزات الأشجار الديناميكية حدًّا زمنيًّا مخمَّدًا $O(\lg n)$ لكل عملية؛ على حين يعطي تنجيزٌ أكثر تعقيدًا حدًّا زمنيًّا $O(\lg n)$ في أسوأ الحالات. تُستخدم الأشجار الديناميكية في بعض أسرع خوارزميات تدفق الشبكات مقاربةً.
- ***أشجار سبلاي Splay trees***، طوَّرها Sleator و Tarjan [320]، وناقشها أيضًا Tarjan [330]، وهي أحد أشكال أشجار البحث الثنائي التي تُنفِّذ عمليات البحث المعيارية بزمن مخمَّد $O(\lg n)$. إن أحد تطبيقات أشجار سبلاي هو تبسيط الأشجار الديناميكية.

- ***بنى المعطيات الدائمة Persistent***، تسمح بالاستعلامات إضافة إلى التحديثات في بعض الأحيان، وذلك على نسخ سابقة من بنى المعطيات. يقدِّم Driscoll و Sarnak و Sleator و Tarjan [98] تقنيات لجعل بنية المعطيات المترابطة دائمة بزمن وحجم قليلين. تعطي المسالة 13-1 مثالاً بسيطًا على مجموعة ديناميكية دائمة.
- وكما في الفصل 20، تسمح عدة بنى معطيات بتنجيز أسرع للعمليات المعجمية (INSERT و DELETE و SEARCH) في حالة فضاء محدود من المفاتيح. يمكن لهذه البنى، بالاستفادة من هذه التحديدات، الوصولُ إلى أزمنة تنفيذ تقريبية، أفضل من بنى المعطيات التي تعتمد على المقارنة، في أسوأ الحالات. أدخَل Fredman و Willard ***أشجار الصهر fusion trees*** [115]، التي كانت أول بنية معطيات تسمح بعمليات معجمية أسرع عندما يكون الفضاء مقتصرًا على الأعداد الصحيحة. وقد بيَّنا كيفية تنجيز هذه العمليات بزمن $O(\lg n/\lg\lg n)$. أعطت عدة بنى معطيات تالية، منها ***أشجار البحث الأسية exponential search trees*** [16] حدودًا محسَّنة على بعض العمليات المعجمية أو كلها، وهي مذكورة في ملاحظات الفصول ضمن الكتاب.
- تدعم ***بنى معطيات البيانات الديناميكية Dynamic graph data structures*** استعلامات مختلفة حينما تسمح بتغيير بنية البيان باستخدام عمليات إدراج وحذف على العقد أو الوصلات. تتضمن أمثلة على الاستعلامات التي تدعمها: ترابط الرؤوس vertex connectivity [166]، وترابط الوصلات edge connectivity، وأشجار المسح الصغرى minimum spanning trees [165]، والترابط الثنائي biconnectivity، والإغلاق المتعدي transitive closure [164].

تَذْكر ملاحظاتُ الفصول في هذا الكتاب بنى معطيات إضافية.

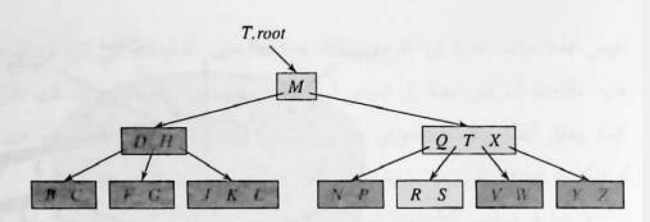
18 الأشجار المعمَّمة

الأشجار المعمَّمة B-trees هي أشجار بحث متوازنة مصمَّمة لتعمل جيدًا على الأقراص أو وسائط الخزن الثانوية الأخرى ذات النفاذ المباشر. تشبه الأشجار المعمَّمة الأشجار الحمراء–السوداء red-black trees (الفصل 13)، لكنها تفضُلها في تقليص عمليات الدخل والخرج I/O على القرص. تَستخدم عدةُ أنظمةِ قواعد معطيات الأشجارَ المعمَّمة أو متغيرات منها، لخزن المعلومات.

تختلف الأشجارُ المعمَّمة عن الأشجار الحمراء–السوداء في أن لعقدها العديد من الأبناء، من بضعة أبناء إلى الآلاف. أي إن عامل التفريع "branching factor" للأشجار المعمَّمة يمكن أن يكون ضخمًا جدًّا، مع أن ذلك يتعلق عادةً بمميزات وحدة القرص المستخدمة. تشبه الأشجار المعمَّمة الأشجار الحمراء–السوداء في أن لكل شجرة معمَّمة من n عقدة ارتفاعاً هو $O(\lg n)$، مع أن الارتفاع الدقيق لشجرة يمكن أن يكون أقل بكثير من ارتفاع شجرة حمراء–سوداء، وذلك لأن عامل التفريع فيها، ومن ثَمَّ قاعدة اللغاريتم التي تعبِّر عن الارتفاع، يمكن أن يكون أكبر بكثير. وبالنتيجة يمكننا استخدام الأشجار المعممة لتنجيز عدة عمليات من عمليات المجموعات الديناميكية بزمن $O(\lg n)$.

إن أشجار B-trees هي التعميم الطبيعي لأشجار البحث الثنائية. يُظهر الشكل 1.18 شجرة معمَّمة بسيطة. إذا تضمنت إحدى العقد الداخلية x في شجرةٍ معممة $x.n$ مفتاحًا، كان لـ x عددٌ من الأبناء يساوي $x.n + 1$. يمكن الاستفادة من المفاتيح في العقدة x كنقاط تقسيمٍ تفصل مجالَ المفاتيح الذي تعالجه x إلى $x.n + 1$ مجالاً جزئيًّا، يُعالِج كلٌّ منها ابنًا من أبناء x. حين نبحث عن مفتاحٍ في شجرة B-tree، فإننا نتخذ قرارًا بـ $x.n + 1$ طريقة، اعتمادًا على $x.n$ مفتاحًا مخزنًا في العقدة x. تختلف بنية العقد في الأوراق عن بنية العقد الداخلية؛ وسندرس هذه الاختلافات في المقطع 1.18.

يُعطي المقطع 1.18 تعريفًا دقيقًا لأشجار B-trees، ونبرهن على أن ارتفاع شجرة B-tree يزداد لغاريتميًّا فقط مع عدد العقد التي تحتويها. ويصف المقطع 2.18 كيفية البحث عن مفتاح، وكيفية إدراج مفتاح في شجرة B-tree. ويناقش المقطع 3.18 عملية الحذف. ولكن، قبل أن نتابع، يجب أن نسأل: لماذا نقيِّم بنى المعطيات المصممة للعمل على الأقراص تقييمًا مختلفًا عن بنى المعطيات المصممة للعمل في الذاكرة الرئيسية العشوائية النفاذ.



الشكل 1.18 شجرة B-tree، مفاتيحها هي الحروف الصوامت في اللغة الإنكليزية. لكل عقدة داخلية x بـ $x.n$ مفتاحًا $x.n + 1$ ابنًا. كل الأوراق على نفس العمق من الشجرة. العقد المظللة تظليلاً خفيفًا هي التي جرى تفحصها للبحث عن الحرف R.

بنى المعطيات في الخزن الثانوي

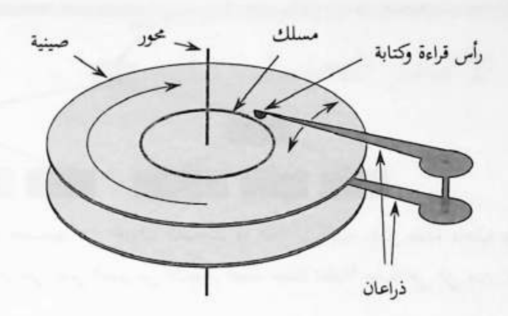
يستفيد نظام حاسوبي من تقانات مختلفة عديدة توفِّر سعات في الذاكرة. تتألف ***الذاكرة الأولية primary memory*** (أو ***الذاكرة الرئيسية main memory***) لنظام حاسوبي من رقائق ذاكرة سيليكونية. إن هذه التقانة، عموماً، أكثر تكلفة لكل بتٍّ مخزَّن من تقانة الخزن المغنطيسي كما في الأشرطة المغنطيسية والأقراص. تحتوي معظم النظم الحاسوبية أيضًا على ***خزن ثانوي secondary storage*** يعتمد على الأقراص المغنطيسية؛ يزيد مقدار هذا الخزن الثانوي غالبًا عن مقدار الذاكرة الرئيسية بمرتبتين على الأقل.

يُظهر الشكل 2.18 سواقة أقراص disk drive نموذجية. تتألف السواقة من ***صينيّة platter*** أو أكثر، تدور بسرعة ثابتة حول ***محور spindle*** مشترك. تغطي سطحَ كل صينية مادةٌ مغنطيسية. تقرأ السواقة كلَّ صينية وتَكتب عليها بواسطة ***رأس head*** يقع في نهاية ***ذراع arm***. يمكن أن تحرِّك الأذرعُ رؤوسَها باتجاه المحور أو بعيدًا عنه. حين يكون رأسٌ ما مستقرًا، يُسمى السطح الذي يمر تحته مباشرةً ***مسلكًا track***. إن تَعدُّد الصينيات يزيد في سعة سواقة القرص فقط، ولا يزيد في أدائها.

ومع أن الأقراص أقل تكلفة وأكبر سعة من الذاكرة الرئيسية، فإنها أبطأ بكثير لأن فيها أجزاء ميكانيكية متحركة.[1] تتألف الحركة الميكانيكية من مكونتين: دوران الصينيات وحركة الذراع. في وقت كتابة هذا الكتاب، كانت الأقراص المتوفرة تدور بسرعات 5400–15000 دورة في الدقيقة (RPM). نرى عادةً سرعات 15000 RPM في السواقات على مستوى المخدِّمات، وسرعات 7200 RPM في سواقات الحواسيب الشخصية، وسرعات 5400 RPM في سواقات الحواسيب المحمولة. ومع أن 7200 RPM قد تبدو سرعة كبيرة، إلاّ أن كل دورة تأخذ 8.33 ميلي ثانية. وهذا تقريبًا، أطول بخمس مرات من 50 نانو ثانية (أكثر أو

---

[1] أثناء كتابة هذا الكتاب وصلت سواقات صلبة الحالة solid-state إلى سوق المستهلك. ومع أنَّ هذه السواقات أسرع من سواقات الأقراص الميكانيكية، إلا أنها تكلِّف أكثر لكل جيغابايت وسعاتها أقل من سعات سواقات الأقراص الميكانيكية.



الشكل 2.18 سواقة قرص نموذجية. تتألف من عدة صينيات تدور حول محور (تظهر هنا صينيتان). تجري القراءة من كل صينية أو الكتابة عليها بواسطة رأس في نهاية ذراع. الأذرع مجموعة معًا بحيث تحرّك رؤوسها بانسجام. تدور الأذرع حول محور دوران مشترك. المسلك هو السطح الذي يمر تحت رأس القراءة أو الكتابة حين يكون مستقرًا.

أقل)، الذي هو زمن النفاذ الشائع إلى ذاكرة السيليكون. وبتعبير آخر، إذا كان علينا الانتظار دورة كاملة لتقع مادةٌ ما تحت رأس القراءة أو الكتابة، أمكننا النفاذ إلى الذاكرة الرئيسية أكثر من 100,000 مرة خلال ذلك المسح. وسطيًا يجب أن ننتظر نصف دورة فقط، لكن يبقى الفرق بين زمن النفاذ إلى ذاكرة السيليكون مقارنةً بزمن النفاذ إلى الأقراص كبيرًا. يأخذ تحريك الذراع أيضًا بعض الوقت. في وقت كتابة هذا النص، تقع أزمنة النفاذ للأقراص المعهودة ضمن المجال 8 إلى 11 ميلي ثانية.

ولكي نخفِّض زمنَ انتظار الحركات الميكانيكية، نُلزِم الأقراصَ بالنفاذ إلى عدة مواد في الوقت نفسه بدلاً من أن تَنْفُذ إلى مادة واحدة فقط. لذا، تُقسَّم المعلومات إلى عدد من ***الصفحات pages*** المتساوية الحجم من البتات، تظهر متتابعةً في الصينيات، وكلُّ قراءةٍ أو كتابةٍ في القرص تكون لصفحة كاملة أو لعدة صفحات. ففي قرص نموذجي، يمكن أن يكون طول الصفحة من 2^{11} إلى 2^{14} بايتًا. بعد أن يأخذ رأس القراءة والكتابة وضعه الصحيح، ويكون القرص قد دار إلى بداية الصفحة المطلوبة، تصبح القراءة على قرص ممغنط أو الكتابة عليه إلكترونية تمامًا (باستثناء دوران القرص)، ويمكن للقرص قراءة مقدار كبير من المعطيات أو كتابتها بسرعة.

في معظم الأحيان، يأخذ وقت النفاذ إلى صفحة من المعلومات وقراءتها من القرص زمنًا أكبر من الزمن الذي يأخذه الحاسوب لفحص كل المعلومات المقروءة. لهذا السبب، سننظر في هذا الفصل إلى كلٍّ من المكونتين الرئيسيتين لزمن التنفيذ على حدة:

- عدد مرات النفاذ إلى القرص، و
- زمن وحدة المعالجة المركزية CPU (زمن الحساب).

نقيس عدد مرات النفاذ إلى القرص بدلالة عدد صفحات المعلومات التي تلزم قراءتها من القرص أو كتابتها عليه. نلاحظ أن زمن النفاذ إلى القرص ليس ثابتًا – بل يتعلق بالمسافة بين المسلك الحالي والمسلك المطلوب، كما يتعلق أيضًا بالموضع الدوراني البدئي للقرص. ومع ذلك، فإننا سنستخدم عدد الصفحات المقروءة أو المكتوبة باعتباره تقريبًا من المرتبة الأولى للزمن الإجمالي المصروف للنفاذ إلى القرص.

في تطبيق نموذجي للأشجار المعمَّمة، يكون حجم المعطيات المعالَجة ضخمًا جدًّا، بحيث لا يمكن وضع كل المعطيات فورًا في الذاكرة الرئيسية. تنسخ خوارزمياتُ الأشجار المعمَّمة صفحاتٍ مختارةً من القرص إلى الذاكرة الرئيسية عند الحاجة إليها، وتعيد كتابة الصفحات التي تغيرت على القرص. تحتفظ خوارزميات B-tree بعددٍ ثابت فقط من الصفحات في الذاكرة الرئيسية في أي وقت؛ لذا لا يَحدّ حجمُ الذاكرة الرئيسية من حجم الأشجار المعمَّمة التي يمكن معالجتها.

ننمذج عمليات القرص في شبه الرماز الخاص بنا كما يلي. ليكن x مؤشرًا إلى غرض. فإذا كان الغرض موجودًا حاليًّا في الذاكرة الرئيسية للحاسوب، أمكننا العودة إلى واصفاته كالمعتاد: على سبيل المثال $x.key$. أما إذا كان الغرض الذي يشير إليه x موجودًا على القرص، فيجب أن ننفِّذ عملية DISK-READ(x) لقراءة الغرض x إلى الذاكرة الرئيسية قبل أن نتمكّن من العودة إلى واصفاته. (نفترض أنه إذا كان x موجودًا سلفًا في الذاكرة الرئيسية، فإن DISK-READ(x) لا يتطلب نفاذًا إلى القرص؛ وهذا يكافئ "لا عملية no-op".) وبالمثل، تُستخدم عملية DISK-WRITE(x) لتخزين أي تغييرات كانت قد جرت على واصفات الغرض x. أي إن الشكل النموذجي للعمل مع غرض ما يكون كالتالي:

x = مؤشر إلى غرض ما

DISK-READ(x)

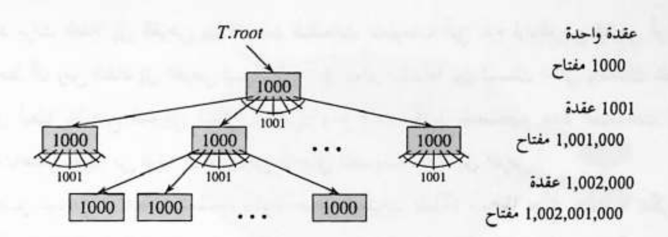
عمليات نفاذ و/أو تعديل على واصفات x

DISK-WRITE(x) // يمكن إغفاله إذا لم يجر أي تعديل على واصفات x

عمليات أخرى تَنْفذ إلى واصفات x، ولكن دون أن تعدّل عليها

يمكن للنظام أن يحتفظ بعدد محدود فقط من الصفحات في الذاكرة الرئيسية في وقت ما. سنفترض أن النظام يُخْرج الصفحات التي لم تعد مستخدمة من الذاكرة الرئيسية؛ وستتجاهل خوارزمياتُ B-tree الخاصة بنا هذا الأمر.

ولمّا كان زمن تنفيذ خوارزمية B-tree، في معظم الأنظمة، يعتمد اعتمادًا رئيسيًّا على عدد عمليات القراءة DISK-READ والكتابة DISK-WRITE التي تجريها على القرص، فإننا نرغب عادةً في أن تقرأ (أو تكتب) كلٌّ من هذه العمليات أكبرَ قدرٍ ممكن من المعلومات. وبذلك، تكون عقدة شجرة B-tree عادةً كبيرة بقدر صفحة قرص كاملة، وهذا الحجم يحد من عدد أبناء عقدة B-tree.



الشكل 3.18 شجرة B-tree ارتفاعها 2 تحتوي على أكثر من مليار مفتاح. يظهر في داخل كل عقدة x عدد المفاتيح $x.n$ في هذه العقدة. تحتوي كل عقدة داخلية وورقة على 1000 مفتاح. ثمة 1001 عقدة على العمق 1، وأكثر من مليون ورقة على العمق 2.

في حالة شجرة B-tree ضخمة مخزَّنة على القرص، يكون عامل التفريع بين 50 و2000 غالبًا، وذلك اعتمادًا على نسبة حجم المفتاح إلى حجم الصفحة. يقلِّص عاملُ التفريع الكبير كلاً من ارتفاع الشجرة وعدد مرات النفاذ إلى القرص المطلوبة للعثور على أي مفتاح تقليصًا كبيرًا. يُظهر الشكل 3.18 شجرة B-tree بعامل تفريع 1001 وارتفاع 2، يمكنها أن تخزِّن أكثر من مليار مفتاح؛ مع ذلك، لما كان بإمكاننا الاحتفاظ دومًا بالعقدة الجذر في الذاكرة، فيمكننا العثور على أي مفتاح في هذه الشجرة بإجراء نفاذين إلى القرص على الأكثر.

1.18 تعريف الأشجار المعمَّمة

سنفترض، للتبسيط، أن أي معلومات تابعة مرفقة مع مفتاحٍ ما ستكون مخزنة في نفس عقدة المفتاح، كما فعلنا في أشجار البحث الثنائية والأشجار الحمراء–السوداء. عمليًّا، يمكننا مع كل مفتاح، تخزينُ مؤشرٍ يشير إلى صفحة قرص أخرى تتضمن المعلومات التابعة لذلك المفتاح. يَفترض شبه الرماز في هذا الفصل ضمنيًّا أنه يجري ترحيل المعلومات التابعة المرفقة مع مفتاح ما، أو المؤشر إلى هذه المعلومات، مع المفتاح فيما إذا تحرك المفتاح من عقدة إلى عقدة. ثمة متغيِّر شائع من الأشجار B-trees، يُعرف بـ ***B^+-trees***، يُخزِّن كل المعلومات التابعة في الأوراق ويخزن فقط المفاتيح ومؤشرات الأبناء في العقد الداخلية، وبذلك يصبح عامل التفريع في العقد الداخلية أعظميًّا.

إن شجرةً معممة T هي شجرة ذات جذر (جذرها هو $T.root$) لها الخصائص التالية:

1. لكل عقدة x فيها الواصفات التالية:

أ. $x.n$ عدد المفاتيح المخزنة حاليًّا في العقدة x،

ب. المفاتيح $x.n$ نفسها، $x.key_1, x.key_2, \dots, x.key_{x.n}$، مخزنة بترتيب غير تنازلي، أي إن:

$x.key_1 \le x.key_2 \le \cdots \le x.key_{x.n}$

ت. $x.leaf$ هي قيمة منطقية: صح TRUE إذا كانت x ورقة، وخطأ FALSE إذا كانت x عقدة داخلية.

2. تحتوي كل عقدة داخلية x أيضًا $x.n + 1$ مؤشرًا $x.c_1, x.c_2, \dots, x.c_{x.n+1}$ إلى أبنائها. وليس للعقد الأوراق أبناء، لذلك فإن واصفات c_i الخاصة بها غير معرَّفة.

3. تَفصل المفاتيحُ $x.key_i$ مجالات المفاتيح المخزنة في كل شجرة جزئية: إذا كان k_i مفتاحًا مخزَّنًا في الشجرة الفرعية ذات الجذر $x.c_i$ فإن:

$k_1 \le x.key_1 \le k_2 \le x.key_2 \le \cdots \le x.key_{x.n} \le k_{x.n+1}$.

4. لكل الأوراق العمق نفسه، وهو ارتفاع الشجرة h.

5. توجد حدود دنيا وحدود عليا لعدد المفاتيح التي يمكن أن تتضمنها عقدة ما. نعبِّر عن هذه الحدود بدلالة عدد صحيح ثابت $t \ge 2$ يُسمَّى ***الدرجة الدنيا minimum degree*** للشجرة B-tree:

أ. يجب أن يكون لكل عقدة (ما عدا عقدة الجذر) $t - 1$ مفتاحًا على الأقل. لذلك، تتضمن كل عقدة داخلية غير الجذر t ابنًا على الأقل. إذا لم تكن الشجرة فارغة، يجب أن يكون للجذر مفتاح واحد على الأقل.

ب. يمكن أن تتضمن كل عقدة $2t - 1$ مفتاحًا على الأكثر. لذلك، يمكن أن يكون لأي عقدة داخلية $2t$ ابنًا على الأكثر. نقول عن عقدة أنها ***ممتلئة full*** إذا كانت تحوي $2t - 1$ مفتاحًا تمامًا.[2]

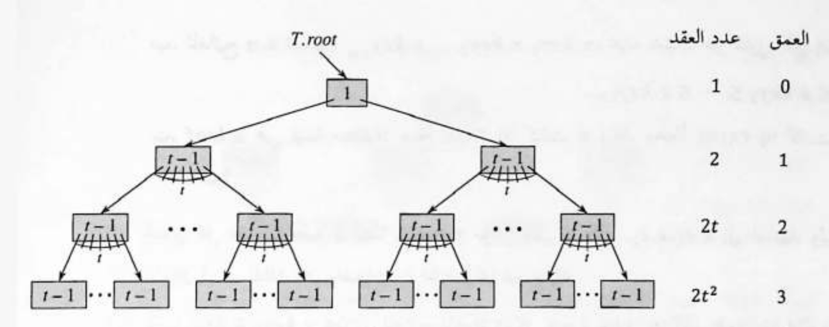
تَحدث أبسط شجرة B-tree عندما يكون $t = 2$. عندئذٍ تحتوي كل عقدة داخلية على ابنَين اثنين، أو 3 أبناء، أو 4 أبناء، ويكون لدينا ***شجرة 2-3-4***. ولكن، عمليًّا تعطي قيم t الكبيرة جدًّا أشجارًا معممة بارتفاعات أقل.

ارتفاع شجرة B-tree

يتناسب عدد مرات النفاذ إلى القرص الذي تتطلبه معظم العمليات على الأشجار B-trees مع ارتفاع الشجرة. نحلِّل الآن ارتفاع شجرة B-tree في أسوأ الأحوال.

---

[2] ثمة متغيّر شائع آخر من الأشجار B-tree، يُعرف بـ ***B^*-tree***، يتطلب أن تكون كل عقدة داخلية ممتلئة بنسبة 2/3 على الأقل، عوضًا عن أن تكون نصف ممتلئة كما تتطلب B-tree.



الشكل 4.18 شجرة معمَّمة ارتفاعها 3، تحتوي على الحد الأدنى من المفاتيح. يظهر ضمن كل عقدة x عدد مفاتيحها $x.n$.

مبرهنة 1.18

إذا كان $n \geq 1$، فيكون لأي شجرة B-tree T ذات n مفتاحًا، وارتفاع h، ودرجة دنيا $t \geq 2$:

$$h \leq \log_t \frac{n+1}{2} .$$

البرهان يتضمن جذر شجرة B-tree T مفتاحًا واحدًا على الأقل، وتتضمن كل العقد الأخرى $t-1$ مفتاحًا على الأقل. إذن، يوجد للشجرة T ذات الارتفاع h، عقدتان على الأقل على عمق 1، و $2t$ عقدة على الأقل على عمق 2، و $2t^2$ عقدة على الأقل على عمق 3، وهكذا... إلى أن نصل إلى العمق h، حيث يوجد $2t^{h-1}$ عقدة على الأقل. يوضح الشكل 4.18 هذه الشجرة في حالة $h = 3$. إذن يحقق العدد n من المفاتيح المتراجحة التالية:

$$\begin{aligned} n &\geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} \\ &= 1 + 2(t-1)\left(\frac{t^h - 1}{t-1}\right) \\ &= 2t^h - 1 . \end{aligned}$$

وبعمليات جبرية بسيطة نحصل على المتراجحة $t^h \leq (n-1)/2$. وبأخذ اللغاريتم ذي الأساس t للطرفين نكون قد برهنا النظرية. ■

نرى هنا قوة الأشجار B-trees مقارنةً بالأشجار الحمراء–السوداء. ومع أن ارتفاع الشجرة يزداد بدرجة $O(\lg n)$ في كلتا الحالتين (تذكَّر أن t ثابت)، إلا أن أساس اللغاريتم في الأشجار المعمَّمة يمكن أن يكون أكبر بعدة مرات. لذلك، توفِّر الأشجار المعمَّمة $\lg t$ تقريبًا على عدد العقد المتفَحَّصة في معظم عمليات

الأشجار، مقارنة بالأشجار الحمراء–السوداء. ولما كان من اللازم عادةً النفاذُ إلى القرص لتفحُّص عقدةٍ ما في شجرة، فإنَّ الأشجار المعمَّمة تتفادى عددًا كبيرًا من مرات النفاذ إلى القرص.

تمارين

1-1.18

لم لا نسمح بدرجة أصغرية $t = 1$؟

2-1.18

ما هي قيمُ t التي تكون فيها الشجرة في الشكل 1.18 شجرةً معمَّمةً صحيحة؟

3-1.18

أظهر جميع الأشجار المعمَّمة الصحيحة من الدرجة الصغرى 2 التي تمثل $\{1, 2, 3, 4, 5\}$.

4-1.18

ما هو العدد الأعظم للمفاتيح التي يمكن تخزينها في شجرة معمَّمة ذات ارتفاع h بدلالة الدرجة الصغرى t؟

5-1.18

صِف بنية المعطيات الناتجة إذا امتصت كل عقدة سوداء في شجرة حمراء–سوداء أبناءها الحمر وضمَّت أبناءها إلى أبنائها.

2.18 العمليات الأساسية على الأشجار المعمَّمة

نقدم في هذا المقطع تفاصيل عمليات البحث B-TREE-SEARCH والإنشاء B-TREE-CREATE والإدخال B-TREE- INSERT على الأشجار المعمَّمة. نعتمد في هذه الإجراءات اصطلاحَين:

- جذر الشجرة المعمَّمة هو دومًا في الذاكرة الرئيسية، لذلك لا نحتاج أبدًا إلى تنفيذ عملية قراءة من القرص DISK-READ للجذر؛ ولكن علينا تنفيذ عملية كتابة على القرص للجذر DISK-WRITE عندما تتغير عقدته.
- يجب تنفيذ عملية قراءة مسبقة DISK-READ لأي عقدة تمرَّر على أنها موسطات.

إن جميع الإجراءات التي نقدمها هي خوارزميات بمرور واحد "one-pass" التي تتقدم نزولاً من جذر الشجرة دون الحاجة إلى الرجوع خلفًا.

البحث في شجرة معمَّمة

يشبه البحث في شجرة معمَّمة كثيرًا البحث في شجرة ثنائية، إلا أننا بدلاً من اتخاذ قرار تفريع ثنائي ذي

طريقين، نتخذ قرارَ تفريع متعدد الطرق بحسب عدد أولاد العقدة. وبتعبير أدق، عند كل عقدة داخلية x، نتخذ قرار تفريع بـ $(x.n+1)$ طريقًا.

إن الإجراء B-TREE-SEARCH هو تعميم مباشر للإجراء TREE-SEARCH المعرَّف على الأشجار الثنائية. دخل الإجراء B-TREE-SEARCH هو المؤشر x إلى عقدة جذر في شجرة جزئية، والمفتاح k الذي يجب البحث عنه في هذه الشجرة الفرعية. لذلك، فإن الاستدعاء على المستوى الأعلى هو من الشكل $\text{B-TREE-SEACH}(T.root, k)$. فإذا كان k موجودًا في الشجرة، فإن هذا الإجراء يعيد الزوجَ المرتب (y, i) الذي يتألف من العقدة y ودليل i بحيث يكون $y.key_i = k$. وإلا، فإنه يعيد القيمة NIL.

```
B-TREE-SEARCH(x, k)
1  i = 1
2  while i ≤ x.n and  k > x.key_i
3      i = i + 1
4  if i ≤ x.n and k == x.key_i
5      return (x, i)
6  elseif x.leaf
7      return NIL
8  else DISK-READ(x.c_i)
9      return B-TREE-SEARCH(x.c_i, k)
```

وباستخدام إجراء بحثٍ خطي، تعثر الأسطر 1–3 على الدليل الأصغر i الذي يحقق $k \le x.key_i$، أو أنها تجعل قيمة i مساوية $x.n+1$. تتفحص الأسطر 4–5 اكتشاف المفتاح، وتعيده إذا اكتشفته. وإلا فإن الأسطر 6–9 تنهي البحث بالإخفاق (إذا كانت x ورقة) أو تطلب تنفيذًا عوديًّا للبحث ضمن الشجرة الفرعية المناسبة لـ x، بعد إجراء القراءة الضرورية من القرص DISK-READ على ذلك الابن.

يُوضِّح الشكل 1.18 عملية البحث B-TREE-SEARCH؛ يتفحص الإجراء العقد المظللة تظليلاً خفيفًا أثناء البحث عن المفتاح R.

كما في حالة الإجراء TREE-SEARCH لأشجار البحث الثنائية، تُشكِّل العقد التي جرت ملاقاتها خلال الإجراء العَوْدي طريقًا بسيطًا نازلاً من جذر الشجرة. ولذا، يَنْفُذ الإجراء B-TREE-SEARCH إلى $O(h) = O(\log_t n)$ صفحة من القرص، حيث h هو ارتفاع الشجرة المعمَّمة و n هو عدد المفاتيح فيها. ولما كان $x.n < 2t$، فإنَّ حلقة **While** في السطرين 2–3 تأخذ زمنًا ضمن كل عقدة هو $O(t)$، ويكون الزمن الإجمالي لوحدة المعالجة المركزية هو $O(t\,h) = O(t \log_t n)$.

إنشاء شجرة معمَّمة فارغة

لبناء شجرة معمَّمة T، نستخدم أولاً B-TREE-CREATE لإنشاء عقدة جذر فارغ، ثم نستدعي

B-TREE-INSERT لإضافة مفاتيح جديدة. يستخدم كلٌّ من هذين الإجراءين إجراءً مساعدًا ALLOCATE-NODE يُخصِّص صفحةً واحدةً في القرص لاستخدامها كعقدة جديدة في زمن $O(1)$. يمكن أن نفترض أن العقدة المنشأة باستخدام ALLOCATE-NODE لا تتطلب قراءة DISK-READ، لأن القرص لا يتضمن بعدُ معلوماتٍ مفيدةً مخزَّنةً على القرص عن هذه العقدة.

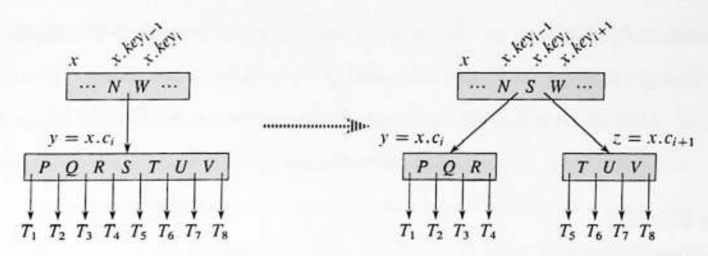
```
B-TREE-CREATE(T)
1  x = ALLOCATE-NODE()
2  x.leaf = TRUE
3  x.n = 0
4  DISK-WRITE(x)
5  T.root = x
```

إن الإجراء B-TREE-CREATE يتطلب $O(1)$ عملية على القرص وزمنًا $O(1)$ من وحدة المعالجة المركزية CPU.

إدراج مفتاح في شجرة معمَّمة

إنَّ إدراج مفتاح في شجرة معمَّمة هو أعقد بكثير من إدراج مفتاح في شجرة بحث ثنائية. كما في حالة أشجار البحث الثنائية، نبحث عن موضع الورقة التي يجب إدراج المفتاح الجديد فيها. لكننا، في الشجرة المعمَّمة لا نستطيع ببساطة إنشاء عقدة ورقة جديدة وإدراجها، لأن الشجرة الناتجة عندها قد تخفق في أن تكون شجرة معمَّمة صالحة. عوضًا عن ذلك، فإننا ندرج المفتاح الجديد في عقدة ورقة موجودة. ولما كنا لا نستطيع إدراج مفتاح في عقدة ورقة ملآنة، فإننا نُدخل عملية ***تفريق splitting*** لعقدةٍ ملآنة y (تتضمن $2t-1$ مفتاحًا) حول ***مفتاحها الوسط median key*** $y.key_t$ إلى عقدتين تتضمن كل منهما $t-1$ مفتاحًا. يتحرك المفتاح الوسط إلى الأعلى ضمن العقدة الأب لـ y ليحدد نقطة التقسيم بين الشجرتين الجديدتين. ولكن، إذا كانت العقدةُ الأب الخاصة بـ y ملآنةً أيضًا، فيجب أن نفرِّقها قبل أن نتمكن من إدراج المفتاح الجديد، ومن ثم فإننا نستطيع إنهاء تفريق العقد الملآنة على كامل الطريق إلى أعلى الشجرة.

وكما في شجرة البحث الثنائية، يمكننا إدراج مفتاح في شجرة معمَّمة بمرور واحد نزولاً ضمن الشجرة من الجذر إلى ورقة. ولعمل ذلك لا ننتظر لمعرفة حاجتنا الفعلية لتفريق عقدة ملآنة لكي ننفِّذ الإدراج. بل نقوم – أثناء انتقالنا نزولاً في الشجرة باحثين عن الموضع الذي ينتمي إليه المفتاح الجديد – بتفريق كل عقدة ملآنة نصل إليها في الطريق (ومن ضمنها الورقة نفسها). وبذلك عندما نريد تفريق عقدة ملآنة y، نكون على يقينٍ بأن العقدة الأب الخاصة بها غير ملآنة.



الشكل 5.18 تفريق عقدة في حالة $t = 4$. جرى تفريق العقدة $y = x.c_i$ إلى عقدتين y و z، وانتقال المفتاح الوسط S في y إلى الأعلى ضمن العقدة الأب.

تفريق عقدة في شجرة معمَّمة

دَخْلُ الإجراء B-TREE-SPLIT-CHILD هو عقدة داخلية *غير ممتلئة* x (نفترض أنها في الذاكرة الرئيسية)، ودليلٌ i، وعقدة y (نفترض أنها في الذاكرة الرئيسية أيضًا) بحيث تكون $x.c_i$ ابنًا *ممتلئًا* لـ x. بعد ذلك، يقوم الإجراء بتفريق هذا الابن إلى اثنين، وضبْطِ x بحيث يصبح له ابنٌ إضافي. ولتفريق جذر ملآن، نجعل هذا الجذر أولاً ابنًا لعقدة جذر جديدة فارغة بحيث نستطيع استخدام B-TREE-SPLIT-CHILD. ومن ثم فإن الشجرة يزيد ارتفاعها بمقدار واحد؛ فالتفريق هو الطريقة الوحيدة التي تنمو بها الشجرة.

يوضِّح الشكل 5.18 هذا الإجراء. نفرِّق العقدة الملآنة $y = x.c_i$ حول مفتاحها الوسط S الذي ينتقل إلى الأعلى إلى العقدة الأب x. جرى وضع المفاتيح الموجودة في y ذات القيمة التي هي أكبر من الوسط ضمن عقدة جديدة z، التي أصبحت ابنًا جديدًا لـ x.

```
B-TREE-SPLIT-CHILD(x, i)
 1  z = ALLOCATE-NODE()
 2  y = x.c_i
 3  z.leaf = y.leaf
 4  z.n = t − 1
 5  for j = 1 to t − 1
 6       z.key_j = y.key_{j+t}
 7  if not y.leaf
 8       for j = 1 to t
 9           z.c_j = y.c_{j+t}
10  y.n = t − 1
11  for j = x.n + 1 downto i + 1
12       x.c_{j+1} = x.c_j
13  x.c_i + 1 = z
14  for j = x.n downto i
```

```
15        x.key_{j+1} = x.key_j
16    x.key_i = y.key_t
17    x.n = x.n + 1
18    DISK-WRITE(y)
19    DISK-WRITE(z)
20    DISK-WRITE(x)
```

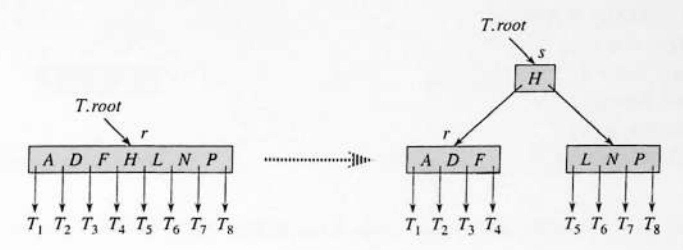
يعمل الإجراء B-TREE-SPLIT-CHILD مباشرة بطريقة القص واللصق. العقدة x هنا هي العقدة التي يجري تفريقها، والعقدة y هي الابن i للعقدة x (وُضِعت في السطر 2). تتضمن العقدة y أصلاً $2t$ ابنًا ($2t-1$ مفتاحًا) ولكن جرى تقليصها بهذه العملية إلى t ابنًا ($t-1$ مفتاحًا). تأخذ العقدة z الأبناء t الكبار ($t-1$ مفتاحًا) في y، وتصبح ابنًا جديدًا لـ x، وتوضع مباشرة بعد y في جدول أبناء x. ينتقل المفتاح الوسط في y إلى الأعلى ليصبح المفتاح الذي يفصل بين y و z في x.

تُنشئ الأسطر 1–9 عقدة z وتعطيها المفاتيح $t-1$ الأكبر والأولاد الموافقين لها في y. يصحح السطر 10 عدد المفاتيح في y. أخيرًا تُدرج الأسطر 11–17 العقدةَ z باعتبارها ابنًا جديدًا لـ x، وتنقل المفتاح الوسط من y إلى x بهدف فصل y عن z، ثم تُصحِّح عدد المفاتيح في x. الأسطر 18–20 تعيد كتابة صفحات القرص المعدَّلة. إن زمن وحدة المعالجة المركزية الذي يستخدمه هذا الإجراء B-TREE-SPLIT-CHILD هو $\Theta(t)$، بسبب الحلقات في الأسطر 5– 6 و 8–9. (الحلقات الأخرى يجري تنفيذها بـ $O(t)$ تكراراً.) يقوم الإجراء بـ $O(1)$ عملية على القرص.

إدخال مفتاح في شجرة معمَّمة بمرورٍ واحد نزولاً عبر الشجرة

نُدرج مفتاحًا k في شجرة معمَّمة T ذات ارتفاع h بمرورٍ واحد نزولاً عبر الشجرة، وهذا يتطلب $O(h)$ نفاذًا إلى القرص. أما وحدة المعالجة المركزية، فتتطلب زمنًا $O(t\,h) = O(t \log_t n)$. ويَستخدم الإجراءُ B-TREE-INSERT الإجراءَ B_TREE_SPLIT_CHILD لضمان عدم نزول العَوْدية إلى عقدة ممتلئة.

```
B-TREE-INSERT(T, k)
 1  r = T.root
 2  if r.n == 2t - 1
 3        s = ALLOCATE-NODE()
 4        T.root = s
 5        s.leaf = FALSE
 6        s.n = 0
 7        s.c_1 = r
 8        B-TREE-SPLIT-CHILD(s, 1)
 9        B-TREE-INSERT-NONFULL(s, k)
10  else B-TREE-INSERT-NONFULL(r, k)
```



الشكل 6.18 تفريق الجذر في حالة $t = 4$. يتفرَّق الجذر r إلى عقدتين، وأُنشئ جذرٌ جديد s. يحتوي الجذر الجديد المفتاح الوسط في r ويحتوي نصفَي r بوصفهما ولدَين. تنمو الشجرة المعمَّمة بزيادة ارتفاعها بمقدار واحد عند تفريق الجذر.

تُعالِج الأسطرُ 3–9 الحالةَ التي يكون فيها الجذر r ممتلئًا: يتفرَّق الجذر، وتُنشِئُ عقدةٌ جديدةٌ s (لها ابنان) هذا الجذر. إن تفريق الجذر هي الطريقة الوحيدة لزيادة ارتفاع الشجرة المعممة. يُظهر الشكل 6.18 هذه الحالة. وخلافًا لشجرة البحث الثنائية، يزداد ارتفاع الشجرة المعمَّمة في قمتها بدلاً من أسفلها. ينتهي الإجراء باستدعاء الإجراء B-TREE-INSERT-NONFULL للقيام بإدخال مفتاح k في الشجرة ذات الجذر غير الممتلئ. ويقوم الإجراء B-TREE-INSERT-NONFULL بتنفيذ عودي نزولاً في الشجرة حسب الضرورة، وفي كل الأوقات، يضمن ألا تكون العقدة التي يعود إليها ممتلئة باستدعاء B_TREE_SPLIT_CHILD عند الضرورة.

يُدرِج الإجراءُ العودي المساعد B-TREE-INSERT-NONFULL مفتاحًا k ضمن عقدة x يُفترض أن تكون غير ممتلئة عند طلب الإجراء. تَضْمن عملية B-TREE-INSERT والعملية العودية B-TREE-INSERT-NONFULL صحة هذا الافتراض.

```
B-TREE-INSERT-NONFULL(x, k)
 1  i = x.n
 2  if x.leaf
 3      while i ≥ 1 and k < x.key_i
 4          x.key_{i+1} = x.key_i
 5          i = i − 1
 6      x.key_{i+1} = k
 7      x.n = x.n + 1
 8      DISK-WRITE(x)
 9  else while i ≥ 1 and k < x.key_i
10          i = i − 1
11      i = i + 1
12      DISK-READ(x.c_i)
```

```
13        if x.c_i.n == 2t − 1
14            B-TREE-SPLIT-CHILD(x, i)
15                if k > x.key_i
16                    i = i + 1
17        B-TREE-INSERT-NONFULL(x.c_i, k)
```

يعمل إجراء B-TREE-INSERT-NONFULL كما يلي. تعالِج الأسطرُ 3–8 الحالةَ التي تكون فيها العقدة x ورقة بإدراج المفتاح k ضمن x. فإذا لم تكن x ورقة، فيجب إدراج k في الورقة المناسبة من الشجرة الفرعية التي يكون جذرها العقدة الداخلية x. في هذه الحالة، تُحدِّد الأسطر 9–11 ابنَ x الذي ينزل إليه التنفيذ العودي. يتحرَّى السطر 13 نزول التنفيذ العودي إلى عقدة ممتلئة، وفي هذه الحالة يَستخدم السطرُ 14 الإجراءَ B-TREE-SPLIT-CHILD لتفريق الابن إلى ابنين غير ممتلئين، ويُحدِّد السطران 15–16 أيَّ الابنين هو الآن العقدة الصحيحة التي يجب النزول إليها. (لاحظ أنه لا حاجة للإجراء DISK-READ$(x.c_i)$ بعد أن يزيد السطر 16 قيمةَ i بواحد، لأن الإجراء العودي سينزل في هذه الحالة إلى ابن أنشأه للتو B-TREE-SPLIT-CHILD.) إن الأثر الصافي الناتج عن الأسطر 13–16 هو إذن ضمان عدم وصول التنفيذ العودي أبدًا إلى عقدة ممتلئة. يقوم السطر 17 بعد ذلك بتنفيذ عودي لإدراج k في الشجرة الفرعية المناسبة. يوضِّح الشكل 7.18 حالات الإدراج المختلفة في شجرة معمَّمة.

يقوم الإجراء B-TREE-INSERT بـ $O(h)$ نفاذًا إلى القرص في حالة شجرة معمَّمة بارتفاع h، لأنه جرى تنفيذ عمليتَي DISK-READ و DISK-WRITE $O(1)$ فقط بين استدعاءات B-TREE-INSERT-NONFULL. أما زمن وحدة المعالجة المركزية الإجمالي المستخدم، فهو $O(t\,h) = O(t \log_t n)$. ولما كان B-TREE-INSERT-NONFULL عَوْدِيَّ الذيل tail-recursive، فيمكن تنجيزه تنجيزًا بديلاً باستخدام حلقة **while**، وهذا يبرهن أن عدد الصفحات التي يلزم بقاؤها في الذاكرة الرئيسية في أي وقت هي $O(1)$.

تمارين

1-2.18

أظهر نتائج إدراج المفاتيح

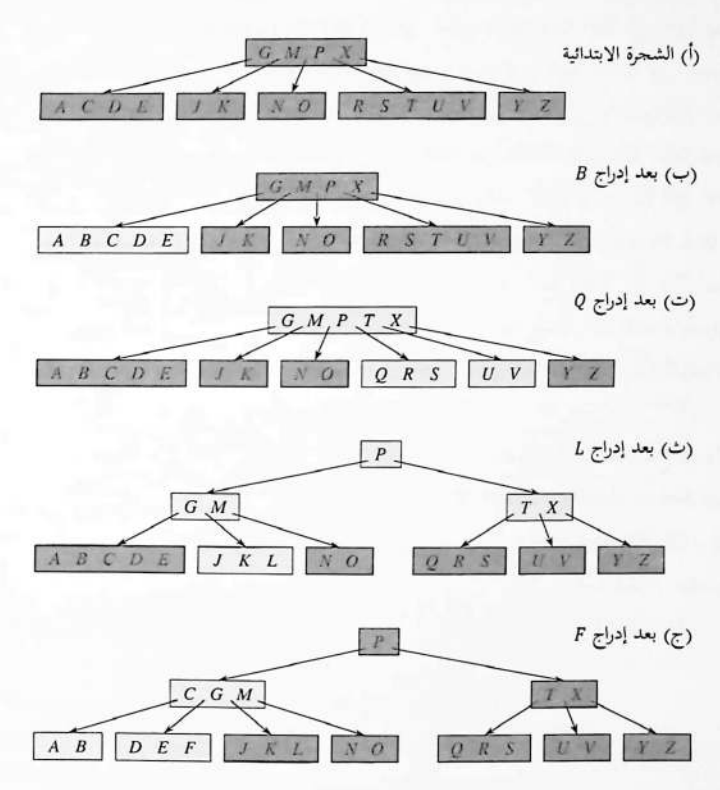
$F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E$

بالترتيب من اليسار إلى اليمين في شجرة معمَّمة فارغة درجتها الصغرى 2. ارسم شكل الشجرة قبل وجوب تفريق عقدة ما فقط، وارسم كذلك شكلها النهائي.

2-2.18

وضّح تحت أية ظروف، إن وجدت، تكون عمليات DISK-READ و DISK-WRITE حشوًا خلال تنفيذ استدعاء للإجراء B-TREE-INSERT. (تكون عملية DISK-READ حشوًا إذا جرت لصفحة موجودة سلفًا

في الذاكرة. وتكون عملية DISK-WRITE حشوًا إذا جرت كتابة صفحة من المعلومات على القرص وكانت هذه الصفحة مطابقةً لما هو مخزّن فيها سابقًا.)



الشكل 7.18 إدراج مفاتيح في شجرة معمّمة. الدرجة الصغرى t في هذه الشجرة المعمّمة هي 3، لذلك يمكن أن تحتوي عقدةٌ ما 5 مفاتيح على الأكثر. العقد المظللة تظليلاً خفيفًا هي العقد التي عدّلها إجراء الإدراج. **(أ)** الشجرة الابتدائية في هذا المثال. **(ب)** نتيجة إدراج B في الشجرة الابتدائية؛ هذا إدراج بسيط في ورقة. **(ت)** نتيجة إدراج Q في الشجرة السابقة. تفرّقت العقدة $RSTUV$ إلى عقدتين تحتويان RS و UV، ونُقِلَ المفتاح T إلى الجذر في الأعلى، وأُدرج Q في النصف الأيسر (العقدة RS). **(ث)** نتيجة إدراج L في الشجرة السابقة. تفرّق الجذر مباشرةً لأنه ممتلئ، وزاد ارتفاع الشجرة بمقدار واحد. ثم أُدرج L في الورقة التي تحتوي JK. **(ج)** نتيجة إدراج F في الشجرة السابقة. تفرّقت العقدة $ABCDE$ قبل إدراج F في النصف الأيمن (العقدة DE).

3-2.18

اشرح كيف يمكن إيجاد المفتاح الأصغر المخزَّن في شجرة معمَّمة، وكيف يمكن إيجاد المفتاح السابق لمفتاح معين مخزَّن في شجرة معمَّمة.

★ ***4-2.18***

افترض أننا أدرجنا المفاتيح $\{1, 2, \ldots, n\}$ في شجرة معمَّمة فارغة درجتها الصغرى 2. ما عدد العقد في الشجرة المعممة النهائية؟

5-2.18

لما كانت العقد الأوراق لا تتطلب مؤشرات إلى الأبناء، فيمكن لها من حيث المبدأ استخدام قيمة مختلفة لـ t (أكبر منها) عن تلك الخاصة بالعقد الداخلية في حالة الحجم نفسه من صفحة القرص. بيّن كيف يمكن تعديل إجراءات الإنشاء والإدراج في شجرة معمَّمة لمعالجة هذا الاختلاف.

6-2.18

افترض أننا نريد تنجيز B-TREE-SEARCH باستخدام بحث ثنائي بدلاً عن البحث الخطي ضمن كل عقدة. بيّن أن هذا التغيير يجعل الزمن المطلوب من وحدة المعالجة المركزية $O(\lg n)$، بمعزل عن كيفية اختيار t باعتباره دالةً لـ n.

7-2.18

افترض أن عتاديات القرص تسمح لنا باختيار حجم صفحة القرص بحرية، لكن الزمن اللازم لقراءة صفحة من القرص هو $a + bt$، حيث a و b ثابتان محدَّدان و t هي الدرجة الصغرى لشجرة معمَّمة تستخدم صفحات من الحجم المختار. صف كيفية اختيار t بحيث يكون زمن البحث في الشجرة المعمَّمة أصغريًّا (تقريبًا). اقترح قيمة أمثلية لـ t في الحالة التي تكون فيها $a = 5$ ميلي ثانية و $b = 10$ ميلي ثانية.

3.18 حذف مفتاح من شجرة معمَّمة

يشبه الحذف من شجرة معمَّمة الإدراج فيها، ولكنه أعقد بقليل، لأنه يمكن حذف مفتاح من أية عقدة – وليس من الأوراق فقط – ويتطلب الحذف من عقدة داخلية إعادة ترتيب أولادها. وكما في حالة الإدراج، يجب أن تحترس من الحذف الذي ينتج عنه شجرة تهتك بنيتُها خواصَّ الأشجار المعمَّمة. ومثلما كان علينا ضمان عدم تضخم عقدةٍ ما تضخمًا كبيرًا بسبب الإدراج، علينا ضمان ألا تصبح عقدةٌ ما صغيرة جدًا بسبب الحذف (باستثناء أن يسمح للجذر بأن يتضمن أقل من العدد الأصغر للمفاتيح $t - 1$.) و كما أن خوارزمية الإدراج البسيطة يمكن أن تتراجع إذا كانت إحدى العقد على الطريق إلى المكان الذي سيجري إدراج المفتاح فيه ملآنة، فإنَّ النهج البسيط للحذف يمكن أن يتراجع إذا كانت عقدة ما (باستثناء الجذر)

على الطريق إلى المكان الذي سيجري حذف المفتاح منه تحتوي على الحد الأدنى من المفاتيح.

يحذف الإجراء B-TREE-DELETE المفتاح k من الشجرة الفرعية ذات الجذر x. نصمِّم هذا الإجراء ليضمن أنَّه كلما استُدعي عوديًّا على عقدة x، فإنَّ عدد المفاتيح في x يساوي على الأقل الدرجة الصغرى t. لاحظ أن هذا الشرط يتطلب مفتاحًا إضافيًّا على عدد المفاتيح الأدنى المطلوب في الشروط المعتادة للشجرة المعمَّمة، لذلك قد يلزم أحيانًا نقل مفتاح ما إلى عقدة ابن قبل نزول التنفيذ العودي إليها. يسمح لنا هذا الشرط المقوّى بحذف مفتاح من الشجرة بمرورٍ واحد نزولاً، دون الحاجة إلى التراجع (باستثناء تراجع واحد سنشرحه). يجب تفسير التوصيف الآتي للحذف من شجرة معمَّمة علمًا بأنه إذا أصبح الجذر x عقدة داخلية بدون مفاتيح (يحصل ذلك في الحالتين 2.ت و 3.ب الآتيتين)، حُذف x وأصبح $x.c_1$، الابن الوحيد لـ x، جذرًا جديدًا للشجرة، وهذا يُنقص ارتفاع الشجرة بمقدار واحد ويحافظ على خاصية أن يتضمن جذر الشجرة مفتاحًا واحدًا على الأقل (إلا إذا كانت الشجرة فارغة).

سنستعرض كيفية عمل الحذف بدلاً من عرض شبه الرماز. يوضِّح الشكل 8.18 الحالات المختلفة لحذف مفاتيح من شجرة معمَّمة.

1. إذا كان المفتاح k في عقدة x وكانت x ورقة، فاحذف المفتاح k من x.
2. إذا كان المفتاح k في عقدة x وكانت x عقدة داخلية، فافعل الآتي.

أ. إذا كان للابن y الذي يسبق k في العقدة x، t مفتاحًا على الأقل، فأوجد المفتاح السابق لـ k وهو k' في الشجرة الفرعية ذات الجذر y. احذف k' عوديًّا، واستعض عن k بـ k' في x. (يمكننا إيجاد k' وحذفه في مرور وحيد نزولاً.)

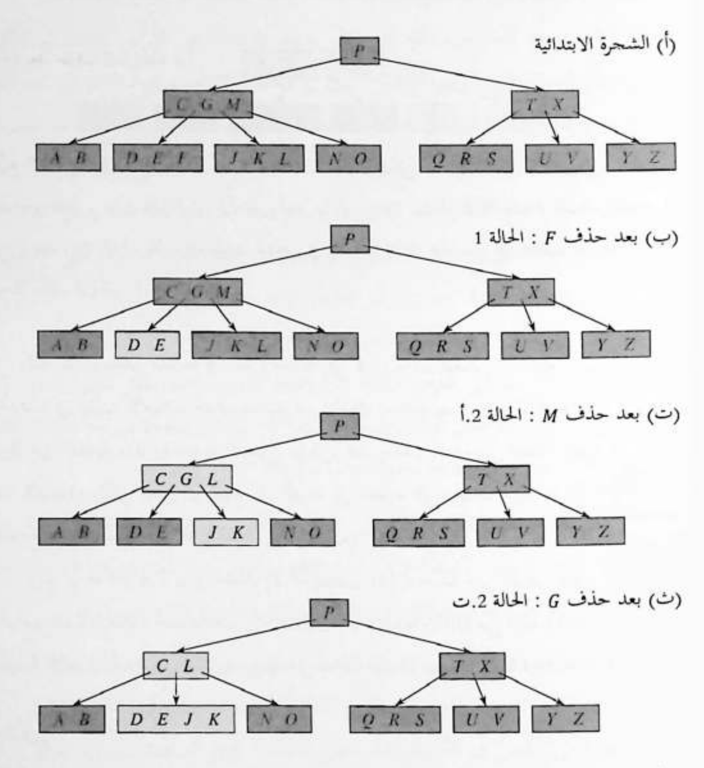
ب. بالتناظر، إذا كان للابن y عددٌ من المفاتيح أقل من t، فتتبَّع الابن z الذي يتبع k في العقدة x، إذا كان الابن z له t مفتاحًا على الأقل، فأوجد المفتاح التالي لـ k وهو k' في الشجرة الفرعية ذات الجذر z. احذف k' عوديًّا، واستعض عن k بـ k' في x. (يمكننا إيجاد k' وحذفه في مرور وحيد نزولاً.)

ت. فيما عدا ذلك، إذا كان لدى كل من y و z فقط $t-1$ مفتاحًا، فادمج k وكل مفاتيح z في y، بحيث تفقد x كلاً من k والمؤشر إلى z. تتضمن y الآن $2t-1$ مفتاحًا. ثم حرِّر z واحذف k عوديًّا من y.

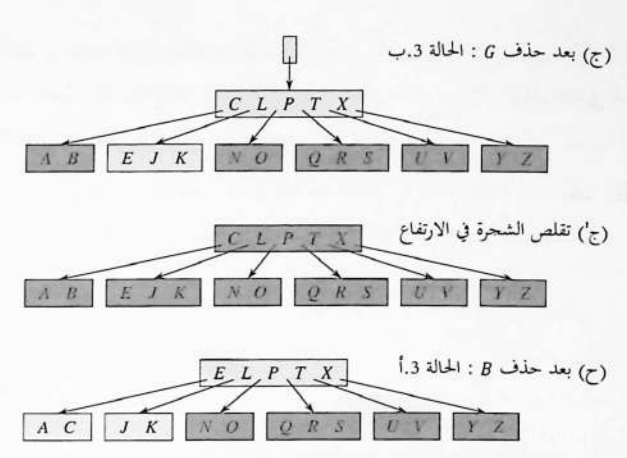
3. إذا لم يكن المفتاح k موجودًا في عقدة داخلية x، حدِّد الجذر $x.c_i$ من الشجرة الفرعية المناسبة الذي يجب أن يحتوي k، إذا كان k موجودًا في الشجرة أصلاً. إذا احتوى $x.c_i$ فقط $t-1$ مفتاحًا، فنفِّذ الخطوة 3.أ أو 3.ب بحسب الضرورة لنضمن أننا ننزل إلى عقدة تحتوي t مفتاحًا على الأقل. ثم ننهي بتنفيذ عودي على الابن المناسب لـ x.

أ. إذا كان لـ $x.c_i$، $t-1$ مفتاحًا، لكن له أخ مباشر يتضمن t مفتاحًا على الأقل، فأعط $x.c_i$ مفتاحًا إضافيًّا بنقل مفتاح من x هبوطًا إلى $x.c_i$، ونقل مفتاح من الأخ المباشر اليميني أو اليساري لـ $x.c_i$ إلى x، ونقل مؤشر الابن المناسب من الأخ إلى $x.c_i$.

ب. إذا تضمن كل من $x.c_i$ وأخويه $t-1$ مفتاحًا، فادمج $x.c_i$ مع أحد إخوته، وهذا يتطلب نقل مفتاح من x إلى الأسفل في العقدة الجديدة المدمجة ليصبح المفتاح الوسط لهذه العقدة.



الشكل 18.8 حذف مفاتيح من شجرة معمَّمة. الدرجة الصغرى لهذه الشجرة المعمَّمة $t = 3$، لذلك لا يمكن لأية عقدة غير الجذر أن تحتوي أقل من مفتاحين. العقد المعدَّلة هي المظللة تظليلاً خفيفًا. (أ) الشجرة المعمَّمة في الشكل 7.18(ج). **(ب)** حذف F. هذه الحالة 1: حذف بسيط من ورقة. **(ت)** حذف M. هذه الحالة 2.أ: المفتاح السابق لـ M وهو L نُقل إلى الأعلى ليأخذ مكان M. **(ث)** حذف G. هذه الحالة 2.ت: دُفع G إلى الأسفل ليكوّن العقدة DEGJK، ثم حُذف G من هذه الورقة (الحالة 1).



يَتْبَع الشكل 18.8 (ج) حذف D. هذه الحالة 3.ب: لا يمكن أن ينزل التنفيذ العودي إلى العقدة CL لأن لديها مفتاحَين فقط ، لذلك دُفع P إلى الأسفل ودُمج مع CL و TX ليُشكل $CLPTX$؛ ثم حُذف D من ورقة (الحالة 1). **(ج')** بعد (ج) حُذف الجذر وتقلصت الشجرة في الارتفاع بمقدار واحد. (ح) حذف B. هذه الحالة 3.أ: نُقل C ليملأ مكان B ونُقل E ليملأ مكان C.

لمّا كانت معظم المفاتيح في شجرة معمَّمة هي في الأوراق، فيمكن أن نتوقع عمليًا أن تُستخدم عمليات الحذف، في غالب الأحيان، لحذف مفاتيح من الأوراق. عندئذٍ يعمل الإجراء B-TREE-DELETE بمرور واحد نزولاً عبر الشجرة دون الحاجة إلى الرجوع. ولكن، عند حذف مفتاح من عقدة داخلية، فإنه يقوم بمرور نزولي عبر الشجرة، ولكن يمكن أن يحتاج إلى العودة إلى العقدة التي حُذف منها المفتاح للاستعاضة عن المفتاح بالمفتاح السابق له أو التالي له (الحالتان 2.أ و 2.ب).

ومع أنَّ هذا الإجراء يبدو معقدًا، إلا أنه يتطلب $O(h)$ عملية على القرص فقط، في حالة شجرة معمَّمة ارتفاعها h، لأن هنالك استدعاءات لـ DISK-READ و DISK-WRITE من الرتبة $O(1)$ فقط بين الاستدعاءات العودية للإجراء. أما الزمن المطلوب من وحدة المعالجة المركزية، فهو $O(t\,h) = O(t \log_t n)$.

تمارين

1-3.18

أظهر نتيجة حذف C و P و V بالترتيب من الشجرة في الشكل 8.18(ح).

2-3.18

اكتب شبه الرماز الخاص بالإجراء B-TREE-DELETE.

مسائل

1-18 مكدسات في الخزن الثانوي

لندرس تنجيز مكدس في حاسوب يتضمن مقدارًا صغيرًا نسبيًّا من الذاكرة الرئيسية السريعة ومقدارًا كبيرًا نسبيًّا من الخزن على القرص الأكثر بطئًا. إن عمليات الدفع PUSH والسحب POP فيه تعمل على قيَم بطول كلمة واحدة. يمكن أن ينمو المكدس الذي نأمل دعمه ليصبح أكبر بكثير مما يمكن أن تتسع له الذاكرة، ومن ثم فإن معظمه يجب أن يكون مخزّنًا على القرص.

ثمة تنجيز بسيط للمكدس، لكنه غير فعّال، يحتفظ به كاملاً على القرص. نحتفظ في الذاكرة بمؤشر إلى المكدس، يمثِّل العنوان على القرص للعنصر العلوي في المكدس. إذا كانت قيمة المؤشر p، كان العنصر العلوي هو الكلمة ذات الترتيب p بالمقاس m، في الصفحة $\lfloor p/m \rfloor$ من القرص، حيث m هو عدد الكلمات في الصفحة.

لتنجيز عملية الدفع PUSH، نزيد مؤشر المكدس واحدًا، ونقرأ الصفحة المناسبة من القرص إلى الذاكرة، وننسخ العنصر المراد دفعه إلى الكلمة المناسبة في الصفحة، ونعيد كتابة الصفحة على القرص. أما عملية السحب، فهي عملية مشابهة. ننقص مؤشر المكدس واحدًا، ونقرأ الصفحة المناسبة من القرص، ونعيد قمة (أعلى) المكدس. لا نحتاج إلى إعادة كتابة الصفحة لأنها لم تُعدَّل.

لمّا كانت العمليات على القرص مكلفة نسبيًّا، فإننا نحسب تكلفتين لكل تنجيز: العدد الإجمالي لمرات النفاذ إلى القرص والزمن الإجمالي لوحدة المعالجة المركزية. كل نفاذ إلى صفحة من m كلمة في القرص يكلِّف نفاذًا واحدًا إلى القرص وزمنًا $\Theta(m)$ لوحدة المعالجة المركزية.

أ. بالمقاربة، ما هو عدد مرات النفاذ إلى القرص في أسوأ الحالات في حالة n عملية على المكدس باستخدام هذا التنجيز البسيط؟ ما هو الزمن الذي تستغرقه وحدة المعالجة المركزية في حالة n عملية على المكدس؟ (عبِّر عن الجواب بدلالة m و n في هذا الجزء والأجزاء التالية.)

لندرس الآن تنجيزًا للمكدس نحتفظ فيه بصفحة من المكدس في الذاكرة. (نحتفظ أيضًا بمقدار صغير من الذاكرة للاحتفاظ بأثر الصفحة الموجودة حاليًّا في الذاكرة.) لا يمكننا إجراء عملية على المكدس إلا إذا كانت الصفحة المعنية من القرص في الذاكرة. عند الضرورة، يمكننا كتابة الصفحة الموجودة حاليًّا في الذاكرة إلى القرص وقراءة الصفحة الجديدة من القرص إلى الذاكرة. إذا كانت الصفحة المعنية من القرص موجودة سابقًا في الذاكرة، فهذا لا يتطلّب نفاذًا إلى القرص.

ب. ما هو عدد مرات النفاذ إلى القرص في أسوأ الحالات التي تتطلّبها n عملية دفع PUSH؟ ما هو الزمن الذي تستغرقه وحدة المعالجة المركزية؟

ت. ما هو عدد مرات النفاذ إلى القرص في أسوأ الحالات الذي تتطلّبه n عملية على المكدس؟ ما هو الزمن الذي تستغرقه وحدة المعالجة المركزية؟

افترض أننا الآن ننجّز المكدس بالاحتفاظ بصفحتين في الذاكرة (إضافةً إلى عدد صغير من الكلمات للاحتفاظ بمعلومات الحجز).

ث. صِف كيفية إدارة صفحات المكدس بحيث يكون العدد المستَهلَك لمرات النفاذ إلى القرص لأي عملية على المكدس $O(1/m)$، والزمن المستَهلَك الذي تستغرقه وحدة المعالجة المركزية لأية عملية على المكدس $O(1)$.

2-18 *الضم والتفريق في الأشجار 2-3-4*

تأخذ عملية ***الضم join*** مجموعتين ديناميكيتين S' و S'' وعنصرًا x بحيث يتحقق لكل عنصر $x' \in S'$ و $x'' \in S''$ المتراجحات: $x'.key < x.key < x''.key$. تعيد هذه العملية مجموعةً $S = S' \cup \{x\} \cup S''$. تشبه عملية ***التفريق split*** عملية ضم "عكسي": إذا أُعطينا مجموعة ديناميكية S وعنصرًا $x \in S$، تُنشئ عملية التفريق مجموعة S' تتألف من جميع عناصر $S-\{x\}$ التي مفاتيحها أصغر من $x.key$ ومجموعة S'' تتألف من جميع عناصر $S-\{x\}$ التي مفاتيحها أكبر من $x.key$. نتحرّى في هذه المسألة، كيفية تنجيز هذه العمليات على الأشجار 2-3-4. نفترض للسهولة أن العناصر تتألف من مفاتيح فقط، وأنَّ كل قيم المفاتيح متمايزة.

أ. بيّن كيف يمكن الاحتفاظ بارتفاع الشجرة الفرعية ذات الجذر x كواصفة $x.height$ لكل عقدة x من شجرة 2-3-4. تأكّد أن تنجيزك لا يؤثّر على الأزمنة المقاربة لتنفيذ البحث والإدراج والحذف.

ب. بيّن كيف يمكن تنجيز عملية الضم. في حالة شجرتين 2-3-4 هما T' وT'' ومفتاح k، يجب أن تُنفَّذ عملية الضم بزمن $O(1 + |h' - h''|)$، حيث h' و h'' هما ارتفاعا T' و T'' على الترتيب.

ت. لندرس المسار البسيط p من جذر شجرة 2-3-4 هي T إلى مفتاح ما k، والمجموعة S' المكوّنة من مفاتيح T التي هي أصغر من k، والمجموعة S'' المكونة من مفاتيح T التي هي أكبر من k. بيّن أن p يقسم S' إلى مجموعة من الأشجار $\{T'_0, T'_1, \ldots, T'_m\}$ ومجموعة من المفاتيح $\{k'_1, k'_2, \ldots, k'_m\}$، حيث لكل قيم $i = 1,2, \ldots, m$ يكون لدينا $y < k'_i < z$ في حالة كل المفاتيح $y \in T'_{i-1}$ و $z \in T'_i$. ما هي العلاقة بين ارتفاع الشجرتين T'_{i-1} و T'_i؟ صِف كيف يقسم p المجموعة S'' إلى مجموعتين من الأشجار والمفاتيح.

ث. بيّن كيف يمكن تنجيز عملية التفريق على T. استخدم عملية الضم لتجميع المفاتيح في S' ضمن شجرة

وحيدة 4-3-2 هي T' وتجميع المفاتيح في S'' ضمن شجرة وحيدة 4-3-2 هي T''. يجب أن يكون زمن تنفيذ عملية التفريق $O(\lg n)$، حيث n هو عدد المفاتيح في T. (*تلميح*: يجب أن تكون تكاليف الضم مضغوطة.)

ملاحظات الفصل

تعطي المراجع Knuth [211]، و Aho و Hopcroft و Ullman [5]، و Sedgewick [306] مناقشات إضافية عن أنواع الأشجار المتوازنة والأشجار المعمَّمة B-trees. يزوّد المرجع Comer [75] دراسة شاملة عن الأشجار المعمَّمة. ويناقش Guibas و Sedgewick [155] العلاقات بين الأنواع المختلفة للأشجار المتوازنة، ومنها الأشجار الحمراء–السوداء والأشجار 4-3-2.

في عام 1970 اخترع J. E. Hopcroft الأشجار 3-2، وهي نوع سابق للأشجار المعمَّمة والأشجار 4-3-2، وفيها لكل عقدة داخلية ابنان أو ثلاثة. أدْخَل Bayer و McCreight [35] الأشجار المعمَّمة B-trees في عام 1972؛ ولم يشرحا سبب اختيارهما لهذا الاسم.

درس المرجع Bender و Demaine و Farach-Colton [40] كيفية جعل الأشجار المعمَّمة تعمل جيدًا في وجود تأثيرات هرمية الذاكرة. تعمل خوارزميات ***cache-oblivious*** الخاصة بهم بفعالية عالية دون أن تعرف صراحةً حجم المعطيات المنقولة ضمن هرمية الذاكرة.

19 كومات فيبوناتشي

تفيد بنية معطيات كومات فيبوناتشي فائدةً مضاعفة. الأولى هي أن هذه الكومات تدعم مجموعة من العمليات التي تؤلِّف ما يُعرف بالكومات "القابلة للدمج" "mergeable heap". والثانية هي أن العديد من عمليات كومات فيبوناتشي يُنفَّذ بزمن مخمَّد ثابت، وهو ما يجعل بنية المعطيات هذه مناسبةً جدًّا للتطبيقات التي تستدعي هذه العمليات بتواتر كبير.

الكومات القابلة للدمج

الكومة القابلة للدمج mergeable heap هي أية بنية معطيات تدعم العمليات الخمس الآتية، لكلِّ عنصرٍ منها مفتاح key:

MAKE-HEAP() تُنشئ وتعيد كومة جديدة لا تحتوي أي عنصر.

INSERT(H, x) تُدرج عنصرًا x – مُلِئ مفتاحُهُ سابقًا – ضمن كومة H.

MINIMUM(H) تُعيد مؤشرًا إلى العنصر الذي يكون مفتاحه أصغريًّا ضمن H.

EXTRACT-MIN(H) تحذف العنصرَ ذا المفتاح الأصغري من H وتُعيد مؤشرًا إليه.

UNION(H_1, H_2) تُنشئ كومةً جديدة تتضمن جميع عناصر الكومتين H_1 و H_2 وتعيدها. يجري في هذه العملية "تدمير" الكومتين H_1 و H_2.

إضافةً إلى عمليات الكومات القابلة للدمج المذكورة، تدعم كومات فيبوناتشي أيضًا العمليتين الآتيتين:

DECREASE-KEY(H, x, k) تُسند إلى العنصر x ضمن الكومة H قيمةً جديدة للمفتاح k، يُفترض ألاَّ تكون أكبر من قيمة مفتاحها الحالية.[1]

---

[1] كما أشرنا في مقدمة الباب الخامس، الكومات المفترضة القابلة للدمج هي الكومات الأصغرية القابلة للدمج mergeable min-heaps، ومن ثم فإنَّ العمليات MINIMUM و EXTRACT-MIN و DECREASE-KEY قابلة للتطبيق. يمكننا بالمقابل تعريف ***كومة أعظمية قابلة للدمج mergeable max-heap*** مع العمليات MAXIMUM و EXTRACT-MAX و INCREASE-KEY.

$\text{DELETE}(H, x)$ يحذف العنصر x من الكومة H.

يُبيِّن الجدول في الشكل 1.19 أنه إذا لم تكن ثمة حاجة إلى العملية UNION، فإنَّ الكومات الثنائية المعتادة – كالمستخدمة في الفرز بالكومة (الفصل 6) – تعمل جيدًا تقريبًا. تُنفَّذ العمليات الأخرى في أسوأ الحالات بزمن $O(\lg n)$ على كومة ثنائية. لكن إذا كنا نحتاج إلى دعم العملية UNION، فإنَّ أداء الكومات الثنائية يكون ضعيفًا. عند ضم الصفيفتين اللتين تحملان الكومتين الثنائيتين المراد دمجهما ثم تنفيذ الإجراء BUILD-MIN-HEAP (انظر المقطع 3.6)، فإن العملية UNION تستغرق زمنًا $\Theta(n)$ في أسوأ الحالات.

وبالمقابل، فإنَّ كومات فيبوناتشي لها حدود أفضل لزمن التنفيذ المقارب من الكومات الثنائية للعمليات: INSERT، و UNION، و DECREASE-KEY ولها أزمنة التنفيذ المقارب نفسها لبقية العمليات. وتجدر ملاحظة أنَّ أزمنة التنفيذ لكومات فيبوناتشي في الشكل 1.19 هي حدود أزمنة مُخمَّدة، وليست حدود أزمنة في أسوأ الحالات لكل عملية. تأخذ عملية UNION زمنًا مخمَّدًا ثابتًا في كومات فيبوناتشي أفضل بكثير من زمن الحالة الأسوأ الخطي الذي تتطلبه الكومات الثنائية (طبعًا بافتراض أنَّ الحد الزمني المخمَّد كافٍ).

كومات فيبوناتشي من الناحية النظرية والعملية

من الناحية النظرية، كومات فيبوناتشي مرغوبة بوجه خاص عندما يكون عدد عمليات EXTRACT-MIN و DELETE قليلاً بالنسبة إلى عدد العمليات الأخرى المنجزة. وهذه الحالة تظهر في عدة تطبيقات. فعلى سبيل المثال، قد تستدعي بعضُ خوارزميات مسائل البيان (graph problems) العمليةَ DECREASE-KEY مرةً لكل وصلة edge. في البيانات الكثيفة التي تتضمن وصلات كثيرة، يقدم الزمن المخمَّد $\Theta(1)$

الإجراء	كومة ثنائية (أسوأ الحالات)	كومة فيبوناتشي (مُخمَّدة)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

الشكل 1.19 أزمنة تنفيذ العمليات على تنجيزي الكومات القابلة للدمج. نرمز لعدد العناصر في الكومة (الكومات) عند تطبيق كل عملية بـ n.

لاستدعاء DECRESE-KEY تحسينًا كبيرًا على زمن أسوأ الحالات $\Theta(\lg n)$ في الكومات الثنائية. تعتمد الخوارزميات السريعة لمسائل مثل حساب أشجار المسح الصغرى (الفصل 23) وإيجاد أقصر المسارات من منبع وحيد (الفصل 24) في أساسها على كومات فيبوناتشي.

أما من الناحية العملية، فإن العوامل الثابتة وتعقيد البرمجة تقلّل من الرغبة في استخدام كومات فيبوناتشي نسبةً إلى الكومات الثنائية العادية (أو الكومات من الدرجة K) في معظم التطبيقات، باستثناء بعض التطبيقات التي تدير حجومًا كبيرة من المعطيات. لذلك فإنَّ كومات فيبوناتشي لها غالبًا أهمية نظرية. ولكنْ إذا جرى تطوير بنية معطيات بسيطة لها الحدود الزمنية المخمَّدة لكومات فيبوناتشي نفسها، فسيكون لها استخدامٌ عمليٌّ أيضًا.

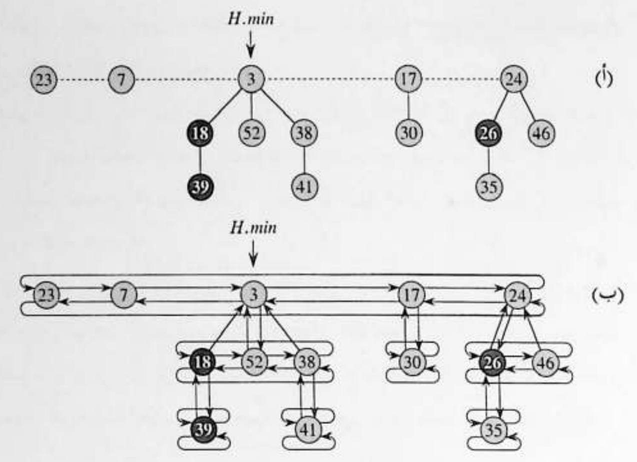
لا تدعم الكومات الثنائية وكومات فيبوناتشي عملية البحث SEARCH على نحو فعال؛ لأن العثور على عقدة تحتوي مفتاحًا معينًا يمكن أن يأخذ وقتًا طويلاً. ولهذا السبب، تتطلب العمليات مثل DECREASE-KEY و DELETE المطبقة على عنصرٍ معين مؤشرًا إلى ذلك العنصر باعتباره جزءًا من دخلها. وكما في مناقشتنا لأرتال الأفضليات في المقطع 5.6، عندما نستخدم كومة قابلة للدمج في تطبيق ما، نُخزِّن غالبًا مقبضًا handle لغرض التطبيق الموافق في كل عنصر من الكومة القابلة للدمج، وكذلك مقبضًا للعنصر الموافق من الكومة القابلة للدمج لكل غرض من أغراض التطبيق. يعتمد تحديد طبيعة هذه المقابض بدقة على التطبيق وتنجيزه.

تعتمد كومات فيبوناتشي، كما رأينا في عدة بنى معطيات أخرى، على الأشجار ذات الجذور. نمثِّل كلَّ عنصر بعقدة ضمن شجرة، وكل عقدة لها واصفة key. سنستخدم التعبير "عقدة" بدلاً من "عنصر" فيما تبقى من هذا الفصل. كما أننا سنتجاهل أمور تخصيص العقد قبل الإدراج وتحريرها بعد الحذف، حيث نفترض أنَّ الرماز الذي يستدعي إجراءات الكومات يعالج هذه التفاصيل.

يُعرِّف المقطع 1.19 كومات فيبوناتشي، ويناقش كيفية تمثيلها، ويقدِّم دالة الكمون المستخدمة في تحليلها المخمَّد. ويُظهر المقطع 2.19 كيفية تنجيز عمليات الكومات القابلة للدمج وكيفية الوصول إلى الحدود الزمنية المخمَّدة الظاهرة في الشكل 1.19. يجري عرض العمليتين المتبقيتين DECREASE-KEY و DELETE في المقطع 3.19. وأخيرًا، ينهي المقطع 4.19 جزءًا هامًا من التحليل ويفسر هذا الاسم الغريب لبنية المعطيات.

1.19 بنية كومات فيبوناتشي

كومة فيبوناتشي Fibonacci heap هي تجمُّعٌ من الأشجار ذات الجذر المرتبة بترتيب ***الكومات وفق الأصغر min-heap ordered***. أي إنَّ كل شجرة تخضع ***لخاصة الكومات وفق الأصغر min-heap property***: مفتاح عقدة ما هو أكبر أو يساوي مفتاح أبيها. يُظهر الشكل 2.19(أ) مثالاً على كومة فيبوناتشي.



الشكل 2.19 (أ) كومة فيبوناتشي مؤلفة من خمسة أشجار مرتبة بترتيب الكومات وفق الأصغر و 14 عقدة. يشير الخط المتقطّع إلى لائحة الجذور. العقدة الصغرى في الكومة هي العقدة التي تحتوي المفتاح 3. جرى تلوين العقد المعلَّمة باللون الأسود. كمون هذه الكومة الخاصة هو $5 + 2 \cdot 3 = 11$. (ب) تمثيل أكثر اكتمالاً يُظهر مؤشرات p (أسهم صاعدة) و $child$ (أسهم هابطة)، و $left$ و $right$ (أسهم جانبية). تُغفِل بقية الأشكال في هذا الفصل هذه التفاصيل، لأنَّ بالإمكان تحديد كل المعلومات المعروضة هنا مما يظهر في الجزء (أ).

وكما يُظهر الشكل 2.19(ب)، تتضمن كل عقدة x مؤشرًا إلى أبيها $x.p$ ومؤشرًا إلى أحد أبنائها $x.child$. يرتبط أبناء x مع بعضهم بلائحة دائرية مضاعفة الترابط، نسميها ***لائحة أبناء x child list***. يتضمن كل ابن y من لائحة الأبناء مؤشرَيْن إلى أخوي y الأيسر والأيمن $y.left$ و $y.right$ على الترتيب. فإذا كانت العقدة y هي الابن الوحيد، يكون $y.left = y.right = y$. يمكن أن يظهر الأشقاء في لائحة الأبناء بأي ترتيب.

إنَّ لاستخدام اللوائح الدائرية المضاعفة الترابط (انظر المقطع 2.10) في كومات فيبوناتشي فائدتَيْن. أولاً، يمكننا إدراج عقدة في أي مكان أو حذف عقدة من أي مكان في لائحة دائرية مضاعفة الترابط بزمن $O(1)$. ثانيًا، إذا كان لدينا لائحتين من هذا النمط، يمكننا ضمهما (أو وصلهما معًا) في لائحة دائرية مضاعفة الترابط بزمن $O(1)$. سنتطرق إلى هذه العمليات شكليًّا عند وصف العمليات على كومات فيبوناتشي، تاركين للقارئ ملء تفاصيل تنجيزها إن رغب بذلك.

تتضمن كل عقدة واصفتان آخران. نخزِّن عدد الأبناء في لائحة الأبناء في عقدة x وهو مخزن في $x.degree$. والواصفة ذو القيمة المنطقية $x.mark$ يشير إلى فقدان العقدة x ابنٍ لها منذ آخر مرة أصبحت فيها ابنًا لعقدة أخرى. تكون العقد المنشأة حديثًا غير معلَّمة، وتصبح أية عقدة x غير معلَّمة

عندما تكون ابنًا لعقدة أخرى. سنكتفي بوضع قيمة FALSE في جميع الواصفات *mark*، إلى أن نتطرق إلى العملية DECREASE-KEY في المقطع 3.19.

يجري النفاذ إلى كومة فيبوناتشي H عن طريق مؤشر $H.min$ إلى جذر شجرة تتضمن مفتاحًا أصغريًّا؛ نُسمي هذه العقدة ***العقدة الصغرى minimum node*** لكومة فيبوناتشي. إذا كان هناك أكثر من جذر له مفتاح بالقيمة الصغرى نفسها، فيمكن أن نعتبر أيًّا منها العقدة الصغرى. إذا كانت كومة فيبوناتشي H فارغة، فإن $H.min =$NIL.

ترتبط جذور جميع الأشجار في كومة فيبوناتشي بعضها ببعض باستخدام مؤشراتها $left$ و $right$ بلائحة دائرية مضاعفة الترابط نسميها ***لائحة جذور root list*** كومة فيبوناتشي. يشير المؤشر $H.min$ إذن إلى العقدة ذات المفتاح الأصغري في لائحة الجذور. يمكن أن تظهر الأشجار في لائحة الجذور بأي ترتيب.

نعتمد على واصفة أخرى في كومة فيبوناتشي H هي $H.n$: العدد الحالي للعقد في H.

دالة الكمون

سنستخدم كما ذكرنا طريقة الكمون المعروضة في المقطع 3.17 لتحليل أداء العمليات على كومات فيبوناتشي. في حالة كومة فيبوناتشي معطاة H، يكون $t(H)$ عدد الأشجار في لائحة جذور H و $m(H)$ عدد العقد المعلَّمة في H. فنعرِّف كمون كومة فيبوناتشي H كالآتي:

$$\Phi(H) = t(H) + 2m(H) \qquad (1.19)$$

(سنعرف المزيد عن دالة الكمون في المقطع 3.19.) فمثلاً، إنَّ كمون كومة فيبوناتشي الظاهرة في الشكل 2.19 هو $5 + 2 \cdot 3 = 11$. وإنَّ كمون مجموعة من كومات فيبوناتشي هو مجموع كمونات كومات فيبوناتشي المؤلِّفة لها. سنفترض أنَّ وحدة الكمون يمكن أن تسدد ثمن مقدار ثابت من العمل كبير كفايةً ليسدد تكلفة أية أجزاء عمل محددة ثابتة الزمن قد نواجهها.

نفترض أنَّ تطبيق كومة فيبوناتشي يبدأ بدون كومات. فيكون الكمون الأولي 0، وبحسب المعادلة 1.19، لا يمكن أن يكون الكمون سالبًا في المرات التالية. وبحسب المعادلة 3.17، يكون الحد الأعلى للتكلفة الكلية المخمَّدة هو حدًّا أعلى للتكلفة الكلية الفعلية لمتتالية العمليات.

الدرجة العظمى

تفترض التحليلات المخمَّدة التي سنجريها في المقاطع المتبقية من هذا الفصل أنَّ هناك حدًّا أعلى معروفًا $D(n)$ للدرجة العظمى لأية عقدة في كومة فيبوناتشي من n عقدة. لن نُثبِتَ ذلك، إلا عندما تُدعم عمليات الكومات القابلة للدمج فقط، $D(n) \leq \lfloor \lg n \rfloor$. (يُطلب إليك في المسألة 19-2(ث) إثبات هذه الخاصة.) سنرى في المقطع 3.19 و 4.19 أنه عندما ندعم DEREASE-KEY و DELETE أيضًا يكون $D(n) = O(\lg n)$.

2.19 عمليات الكومات القابلة للدمج

تُؤخِّر عمليات الكومات القابلة للدمج على كومات فيبوناتشي العملَ قدر المستطاع. فهناك تسوية (مقايضة) للأداء بين تنجيزات العمليات المختلفة. فإذا أدرجنا مثلاً عقدةً بإضافتها إلى لائحة الجذور، فإن هذا يتطلب زمنًا ثابتًا فقط. أما إذا كنا قد بدأنا من كومة فيبوناتشي فارغة، ثم أدرجنا k عقدة، فستتألف الكومة من لائحة جذور ذات k عقدة فقط. وتكون التسوية (المقايضة) بأننا إذا نفَّذنا عملية EXTRACT-MIN على كومة فيبوناتشي H، بعد حذف العقدة التي يشير عليها $H.min$، فعلينا أن ننظر ضمن كل عقدة من العقد $k-1$ المتبقية في لائحة الجذور للعثور على العقدة الصغرى الجديدة. وأثناء مرورنا على كامل لائحة الجذور خلال عملية EXTRACT-MIN فإننا نقوم أيضًا بتدعيم consolidate العقد وتحويلها إلى أشجار كومات مرتبة وفق الأصغر لتقليص حجم لائحة الجذور. سنرى أنه لا يهم شكل لائحة الجذور قبل عملية EXTRACT-MIN، فبعد ذلك سيكون لكل عقدة في لائحة الجذور درجة وحيدة ضمن لائحة الجذور، وهذا يؤدي إلى لائحة جذور حجمها $D(n)+1$.

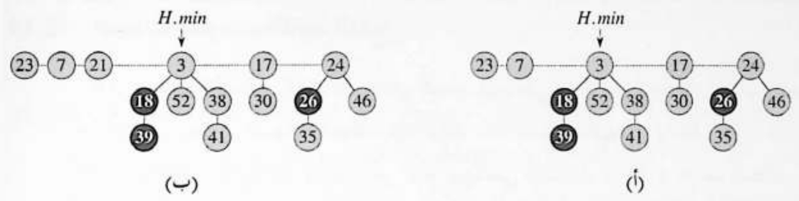
إنشاء كومة فيبوناتشي جديدة

لإنشاء كومة فيبوناتشي فارغة يُخصِّص الإجراء MAKE-FIB-HEAP غرض كومة فيبوناتشي H ويعيده، حيث $H.n = 0$ و $H.min =$NIL؛ أي ليس هناك أشجار في H. ولمّا كان $t(H) = 0$ و $m(H) = 0$ فإنَّ كمون كومة فيبوناتشي الفارغة هو $\Phi(H) = 0$. وبذلك تساوي التكلفة المخمَّدة للإجراء MAKE-FIB-HEAP تكلفته الفعلية $O(1)$.

إدراج عقدة

يقوم الإجراء التالي بإدراج عقدة x في كومة فيبوناتشي H، مفترضًا أنَّ العقدة جرى حصحصتها وأنَّ المفتاح $x.key$ جرى ملؤه سلفًا.

```
FIB-HEAP-INSERT(H, x)
 1  x.degree = 0
 2  x.p = NIL
 3  x.child = NIL
 4  x.mark = FALSE
 5  if H.min == NIL
 6      create a root list for H containing just x
 7      H.min = x
 8  else insert x into H's root list
 9      if x.key < H.min.key
10          H.min = x
11  H.n = H.n + 1
```



الشكل 3.19 إدراج عقدة في كومة فيبوناتشي. **(أ)** كومة فيبوناتشي H. **(ب)** كومة فيبوناتشي H بعد إدراج العقدة ذات المفتاح 21. تصبح العقدة شجرة مرتبة بترتيب الكومة وفق الأصغر ثم تُضاف إلى لائحة الجذور لتصبح الأخ الأيسر للعقدة الصغرى.

تقوم الأسطر 1-4 باستبداء الواصفات البنيوية للعقدة x. يختبر السطر 5 كون كومة فيبوناتشي H فارغة؛ فإذا كانت كذلك يجعل السطران 6-7 العقدةَ x العقدة الوحيدة في لائحة جذور H ويجعلان $H.min$ تشير إلى x. وإلا، فإنَّ الأسطر 8-10 تدرج x ضمن لائحة جذور H تُحدِّث $H.min$ إذا كان ذلك ضروريًا. أخيرًا، يزيد السطر 10 قيمة $H.n$ ليأخذ بذلك إضافة العقدة الجديدة بالاعتبار. يُظهر الشكل 3.19 عقدةً مفتاحها 21 مدرجة في كومة فيبوناتشي الظاهرة في الشكل 2.19.

ولتحديد التكلفة المخمَّدة للإجراء FIB-HEAP-INSERT، نفترض أن H كومة فيبوناتشي المدخلة و H' كومة فيبوناتشي الناتجة. عندها يكون لدينا $t(H') = t(H) + 1$ و $m(H') = m(H)$، وتكون الزيادة في الكمون هي:

$$((t(H) + 1) + 2m(H)) - (t(H) + 2m(H)) = 1 .$$

ولمّا كانت التكلفة الفعلية هي $O(1)$، فإن التكلفة المخمَّدة $O(1) + 1 = O(1)$.

إيجاد العقدة الصغرى

تعطَى العقدة الصغرى في كومة فيبوناتشي H بالمؤشر $H.min$، ومن ثم يمكننا إيجاد العقدة الصغرى بزمن فعلي $O(1)$. ولمّا كان كمون H لا يتغير، فإنَّ التكلفة المخمَّدة لهذه العملية تساوي تكلفتها الفعلية $O(1)$.

توحيد كومتَيْ فيبوناتشي

يوحِّد الإجراء التالي كومتي فيبوناتشي H_1 و H_2 مع تدمير الكومتين الأصليتين خلال الإجرائية. وهو يقوم ببساطة بضم لائحتي جذور H_1 و H_2 ثم يحدد العقدة الصغرى الجديدة. بعد ذلك لن يجري استخدام الأغراض التي تمثل H_1 و H_2.

```
FIB-HEAP-UNION(H1, H2)
1  H = MAKE-FIB-HEAP()
2  H.min = H1.min
```

```
3  concatenate the root list of H_2 with the root list of H
4  if (H_1.min == NIL) or (H_2.min ≠ NIL and H_2.min.key < H_1.min.key)
5      H.min = H_2.min
6  H.n = H_1.n + H_2.n
7  return H
```

تضم الأسطر 1-3 لائحتي جذور H_1 و H_2 في لائحة جذور جديدة H. تحدد الأسطر 2 و4 و5 العقدة الصغرى في H، ويضع السطر 6 العدد الكلي للعقد في $H.n$. يعيد السطر 7 كومة فيبوناتشي الناتجة H. وكما في إجراء FIB-HEAP-INSERT تبقى الجذور كلها جذورًا.

التغيُّر في الكمون هو

$$\begin{aligned}\Phi(H) - (\Phi(H_1) + \Phi(H_2)) \\ &= \big(t(H) + 2m(H)\big) - ((t(H_1) + 2m(H_1)) + (t(H_2) + 2m(H_2))) \\ &= 0\ ,\end{aligned}$$

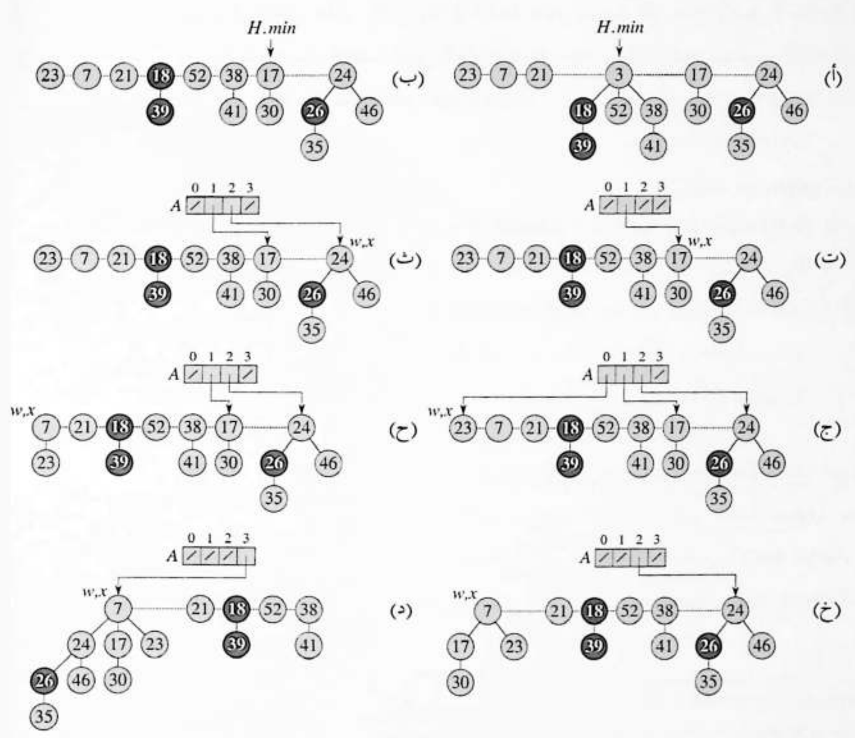
لأن $t(H) = t(H_1) + t(H_2)$ و $m(H) = m(H_1) + m(H_2)$. وبذلك، تكون التكلفة المخمَّدة للإجراء FIB-HEAP-UNION مساويةً تكلفته الفعلية $O(1)$.

استخراج العقدة الصغرى

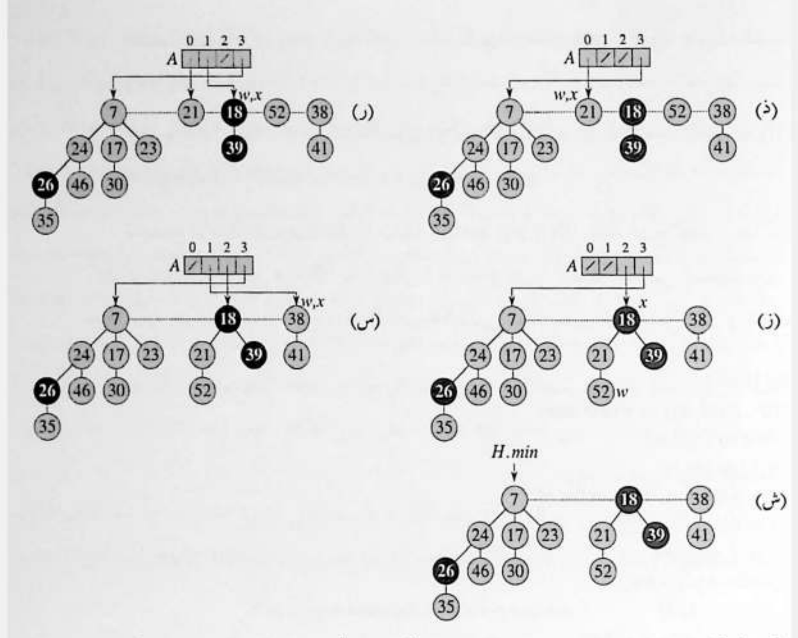
تعتبر إجرائية استخراج العقدة الصغرى أعقد العمليات المعروضة في هذا المقطع. وهي أيضًا العملية التي يُجرى فيها العمل المؤجل لتدعيم الأشجار في لائحة الجذور. يَستخرج شبه الرماز التالي العقدةَ الصغرى. يفترض الرماز اصطلاحًا أنَّ المؤشرات المتبقية في اللائحة المترابطة تُحدَّث عند حذف عقدة من اللائحة المترابطة، لكن تبقى المؤشرات في العقدة المستخرجة دون تغيير. يَستخدم الرماز أيضًا إجراءً مساعدًا CONSOLIDATE سنراه باختصار.

```
FIB-HEAP-EXTRACT-MIN(H)
 1  z = H.min
 2  if z ≠ NIL
 3      for each child x of z
 4          add x to the root list of H
 5          x.p = NIL
 6      remove z from the root list of H
 7      if z == z.right
 8          H.min = NIL
 9      else H.min = z.right
10          CONSOLIDATE(H)
11      H.n = H.n − 1
12  return z
```

وكما يوضِّح الشكل 19-4، يقوم FIB-HEAP-EXTRACT-MIN أولاً بجعل كل عقدة من أبناء العقدة الصغرى جذرًا وحذف العقدة الصغرى من لائحة الجذور. ثم يدعِّم لائحة الجذور بربط الجذور ذات الدرجات المتساوية إلى أن يبقى على الأكثر جذر واحد من كل درجة.



الشكل 19-4 عمل FIB-HEAP-EXTRACT-MIN. (أ) كومة فيبوناتشي H. **(ب)** الوضع بعد حذف العقدة الصغرى z من لائحة الجذور وإضافة أبنائها إلى هذه اللائحة. **(ت)**–**(ج)** الصفيفة A والأشجار بعد كل من التكرارات الثلاثة الأولى لحلقة **for** في الأسطر 4-14 من الإجراء CONSOLIDATE. يعالج الإجراء لائحة الجذور بدءًا من الجذر الذي يشير إليه $H.min$ وباتجاه المؤشرات *اليمينية* $right$. يُظهر كل جزء قيم w و x في نهاية تكرار ما. **(ح)**-**(د)** التكرار التالي للحلقة **for** مع إظهار قيم w و x في نهاية كل تكرار لحلقة **while** في الأسطر 7-13. يُظهر الجزء (ح) الوضع بعد المرور الأول في الحلقة **while**. جرى ربط العقدة ذات المفتاح 23 بالعقدة ذات المفتاح 7، والتي يشير إليها x الآن. في الجزء (خ)، جرى ربط العقدة ذات المفتاح 17 بالعقدة ذات المفتاح 7 التي مازال x يُشير إليها. في الجزء (د)، جرى ربط العقدة ذات المفتاح 24 بالعقدة ذات المفتاح 7. ولما كانت $A[3]$ لا تشير سلفًا إلى أية عقدة في نهاية تكرار حلقة **for**، فيجري وضع المؤشر $A[3]$ على جذر الشجرة الناتجة.



يَتَّبِع الشكل 4.19 (ذ)–(س) الوضع بعد كل من التكرارات الأربعة التالية للحلقة **for**. **(ش)** كومة فيبوناتشي H بعد إعادة بناء لائحة الجذور من الصفيفة A وتحديد المؤشر الجديد $H.min$.

نبدأ من السطر 1 بتخزين مؤشر z إلى العقدة الصغرى؛ يُعيد الإجراء هذا المؤشر في النهاية. إذا كان $z = \text{NIL}$، فإنَّ كومة فيبوناتشي H هي فارغة أصلاً، ونكون قد انتهينا. وإلاَّ نُحذف العقدة z من H، وذلك بجعل جميع أبناء z جذورًا في H في الأسطر 3–5 (بوضعهم في لائحة الجذور)، وحذف z من لائحة الجذور في السطر 6. إذا كان z هو نفسه الأخ الأيمن بعد السطر 6، تكون z هي العقدة الوحيدة في لائحة الجذور ولا يكون لديها أبناء، فكل ما تبقى هو إفراغ كومة فيبوناتشي في السطر 8 قبل إعادة z. إن لم يكن كذلك، نضع المؤشر $H.min$ على لائحة الجذور ليشير إلى جذر مختلف عن z (في هذه الحالة الأخ الأيمن له)، والتي قد لا تكون بالضرورة العقدة الصغرى الجديدة عند انتهاء FIB-HEAP-EXTRACT-MIN. يُظهر الشكل 4.19(ب) كومة فيبوناتشي المعروضة في الشكل 4.19(أ) بعد تنفيذ السطر 9.

الخطوة التالية التي نقلص فيها عدد الأشجار في كومة فيبوناتشي هي ***تدعيم consolidating*** لائحة جذور H؛ التي يقوم بها طلب CONSOLIDATE(H). يكون تدعيم لائحة الجذور بتنفيذ متكرر للخطوات التالية إلى أن يصبح لكل جذر في لائحة الجذور درجة ***degree*** متمايزة.

1. اكتشف جذرين x و y لهما الدرجة نفسها في لائحة الجذور. ودون فقدان العمومية، ليكن $x.key \leq y.key$.

2. *اربط* y بـ x: احذف y من لائحة الجذور، واجعل y ابنًا لـ x بطلب الإجراء FIB-HEAP-LINK. يزيد هذا الإجراء قيمة الواصفة $x.degree$ ويزيل العلامة من y.

يَستخدم الإجراء CONSOLIDATE صفيفةً مساعدة $A[0..D(H.n)]$ لتتبع الجذور وفقًا لدرجاتها. إذا كان $A[i] = y$، فيكون y حاليًّا جذرًا يحقق $y.degree = i$. بهدف تخصيص الصفيفة يجب أن نَعْرف طبعًا كيفية حساب الحد الأعلى $D(H.n)$ للدرجة العظمى، لكننا سنرى كيف نفعل ذلك في المقطع 4.19.

```
CONSOLIDATE(H)
 1  let A[0 .. D(H.n)] be a new array
 2  for i = 0 to D(H.n)
 3      A[i] = NIL
 4  for each node w in the root list of H
 5      x = w
 6      d = x.degree
 7      while A[d] ≠ NIL
 8          y = A[d]          // another node with the same degree as x.
 9          if x.key > y.key
10              exchange x with y
11          FIB-HEAP-LINK(H, y, x)
12          A[d] = NIL
13          d = d + 1
14      A[d] = x
15  H.min = NIL
16  for i = 0 to D(H.n)
17      if A[i] ≠ NIL
18          if H.min == NIL
19              create a root list for H containing just A[i]
20              H.min = A[i]
21          else insert A[i] into H's root list
22              if A[i].key < H.min.key
23                  H.min = A[i]
```

```
FIB-HEAP-LINK(H, y, x)
1  remove y from the root list of H
2  make y a child of x, incrementing x.degree
3  y.mark = FALSE
```

يعمل الإجراء CONSOLIDATE تفصيليًّا كالتالي. تخصّص الأسطر 1-3 الصفيفة A وتجعل قيم كل عنصر فيها NIL. تعالج حلقة **for** في الأسطر 4-14 كل جذر w في لائحة الجذور. أثناء ربط الجذور، يمكن ربط w بعقدة أخرى، ومن ثم لا تبقى جذرًا. ومع ذلك، تبقى w دومًا ضمن شجرة لها جذر ما x الذي قد يكون هو w نفسه أو لا. ولمّا كنا نريد جذرًا واحدًا على الأكثر من كل درجة، فإننا ننظر إلى الصفيفة A لنرى: هل تحتوي على جذر y له درجة x نفسها؟ فإذا كانت كذلك نربط الجذرين x و y، ولكن مع ضمان بقاء x جذرًا بعد الربط. أي نربط y بـ x بعد تبديل مؤشرات الجذرين إذا كان مفتاح y أصغر من مفتاح x. بعد ربط y بـ x، نزيد درجة x بمقدار 1، وهكذا نتابع هذه الإجرائية، بربط x بجذر آخر تتساوى درجته مع درجة x الجديدة، إلى أن لا يبقى هناك جذر من الجذور التي عالجناها له درجة x نفسها. ثم نجعل عنصر A الموافق يشير إلى x، بحيث نكون قد سجلنا أن x هو الجذر الوحيد من درجته الذي عالجناه سلفًا عندما نعالج جذورًا أخرى فيما بعد. عندما تنتهي حلقة **for** هذه، سيبقى على الأكثر جذر واحد من كل درجة، وستشير الصفيفة A إلى كل جذر متبقٍ.

تكرر حلقة **while** في الأسطر 7-13 ربط الجذر x للشجرة التي تحتوي العقدة w بشجرة أخرى لجذرها درجة x نفسها، وذلك إلى أن لا يكون لجذر آخر الدرجة نفسها. تحافظ حلقة **while** هذه على اللامتغير التالي:

في بداية كل تكرار للحلقة **while** يكون $d = x.degree$.

نستخدم لامتغير الحلقة هذا كالتالي:

الاستبداء: يضمن السطر 6 تحقق لامتغير الحلقة أول مرة ندخل فيها الحلقة.

المحافظة: في كل تكرار لحلقة **while**، يشير $A[d]$ إلى جذر ما y. لمّا كان $d = x.degree = y.degree$ فإننا سنربط x بـ y. ومن يملك منهما المفتاح الأصغر يكون أبًا للآخر بعد عملية الربط، ولذلك يبدِّل السطران 9-10 المؤشرَيْن إلى x و y إذا كان ذلك ضروريًّا. ثم نربط y بـ x باستدعاء FIB-HEAP-LINK(H, y, x) في السطر 11. يزيد هذا الاستدعاء من قيمة $x.degree$ لكنه يترك $y.degree$ كما هي d. وحيث إنَّ العقدة y لم تَعُدْ جذرًا، لذا فإن السطر 12 يَحذف المؤشر إليها من الصفيفة A. ولما كان استدعاء FIB-HEAP-LINK يزيد قيمة $x.degree$، فإن السطر 13 يَستعيد اللامتغير $d = x.degree$.

الانتهاء: نكرر الحلقة **while** إلى أن يصبح $A[d] =$ NIL، وفي هذه الحالة لا يكون هناك جذر آخر له درجة x نفسها.

بعد انتهاء حلقة **while** نجعل $A[d]$ يشير إلى x في السطر 14 ونقوم بالتكرار التالي لحلقة **for**.

تُظهر الأشكال 4.19 (ت)–(ج) الصفيفة A والأشجار الناتجة بعد التكرارات الثلاثة الأولى لحلقة **for** في الأسطر 4-14. في التكرار التالي لحلقة **for**، تحصل ثلاث عمليات ربط؛ تَظهر نتائجها في الأشكال 4.19 (ح)–(د). تُظهر الأشكال 4.19(ذ)–(س) نتيجة التكرارات الأربعة التالية لحلقة **for**.

كل ما تبقى هو التنظيف. عندما تنتهي حلقة **for** في الأسطر 4-14، يُفرغ السطر 15 لائحة الجذور، وتعيد الأسطر 16-23 بناءها اعتبارًا من الصفيفة A. تَظهر كومةُ فيبوناتشي الناتجة في الشكل 4.19(ش). بعد تدعيم لائحة الجذور يُنهي FIB-HEAP-EXTRACT-MIN عمله بإنقاص قيمة $H.n$ في السطر 11 وإعادة مؤشرٍ إلى العقدة المحذوفة z في السطر 12.

نبيِّن الآن أنَّ التكلفة المخمَّدة لاستخراج العقدة الصغرى من كومة فيبوناتشي ذات n عقدة هي $O(D(n))$. نرمز بـ H لكومة فيبوناتشي قبل عملية FIB-HEAP-EXTRACT-MIN مباشرةً.

نبدأ بحساب الكلفة الفعلية لاستخراج العقدة الصغرى. تأتي مساهمة الحد $O(D(n))$ من كون FIB-HEAP-EXTRACT-MIN يعالِج $D(n)$ ابنًا على الأكثر من أبناء العقدة الصغرى، ومن العمل في الأسطر 2-3 و 16-23 من CONSOLIDATE. يبقى تحليل مساهمة الحلقة **for** في الأسطر 4-14 من CONSOLIDATE، والتي نستخدم لأجلها تحليلاً مجمَّعًا. إنَّ حجم لائحة الجذور عند طلب CONSOLIDATE هو $D(n) + t(H) - 1$ على الأكثر، لأنها تتألف من عقد لائحة الجذور الأصلية $t(H)$ مطروحًا منها عقدة الجذر المستخرجة، ومضافًا إليها أبناء العقدة المستخرجة الذين لا يتعدى عددهم $D(n)$. ضمن تكرار معين من حلقة **for** في الأسطر 4-14 يعتمد عدد تكرارات الحلقة **while** في الأسطر 7-13 على لائحة الجذور. لكننا نعلم أنَّه في كل مرور في الحلقة **while** يجري ربط أحد الجذور بجذر آخر، وبذلك يكون العدد الكلي لتكرارات حلقة **while** مع كل تكرارات حلقة **for** هو على الأكثر عدد الجذور في لائحة الجذور. ومن ثمَّ يتناسب مقدار العمل الكلي المنفَّذ في حلقة **for** على الأكثر مع $D(n) + t(H)$. فيكون العمل الفعلي الكلي لاستخراج العقدة الصغرى هو $O(D(n) + t(H))$.

إن الكمون قبل استخراج العقدة الصغرى هو $t(H) + 2m(H)$، والكمون بعد ذلك هو $(D(n) + 1) + 2m(H)$ على الأكثر، لأن لدينا $D(n) + 1$ جذرًا متبقيًا على الأكثر، ولم يجرِ تعليم أية عقدة خلال العملية. فتكون التكلفة المخمَّدة على الأكثر

$$
\begin{aligned}
O(D(n) + t(H)) &+ ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \\
&= O(D(n)) + O(t(H)) - t(H) \\
&= O(D(n)) ,
\end{aligned}
$$

لأن بإمكاننا رفع قيمة وحدات الكمون لتطغى على الثابت المضمَّن في $O(t(H))$. حدسيًّا، تسدد تكلفة إجراء كل رابط من تقليص الكمون نظرًا لأن الرابط ينقص عدد الجذور بمقدار واحد. سنرى في المقطع 4.19 أنَّ $D(n) = O(\lg n)$، فتكون التكلفة المخمَّدة لاستخراج العقدة الصغرى $O(\lg n)$.

تمارين

1-2.19

اعرض كومة فيبوناتشي الناتجة عن استدعاء FIB-HEAP-EXTRACT-MIN على كومة فيبوناتشي المعروضة في الشكل 4.19(ش).

3.19 إنقاص قيمة مفتاح وحذف عقدة

في هذا المقطع، نبيِّن كيفية إنقاص قيمة مفتاح عقدة في كومة فيبوناتشي بزمن مخمَّد $O(1)$ وكيفية حذف أية عقدة من كومة فيبوناتشي ذات n عقدة بزمن مخمَّد $O(D(n))$. سنثبت في المقطع 4.19 أنَّ الدرجة العظمى $D(n)$ هي $O(\lg n)$، وهذا يعني أن تنفيذ كلٍّ من FIB-HEAP-EXTRACT-MIN و FIB-HEAP-DELETE يتم بزمنٍ مخمَّد $O(\lg n)$.

إنقاص قيمة مفتاح

في شبه الرماز التالي للعملية FIB-HEAP-DECREASE-KEY، نفترض – كما أسلفنا – أنَّ حذف عقدة من لائحة مترابطة لا يغيِّر أي من الواصفات البنيوية في العقدة المحذوفة.

```
FIB-HEAP-DECREASE-KEY(H, x, k)
1  if k > x.key
2      error "new key is greater than current key"
3  x.key = k
4  y = x.p
5  if y ≠ NIL and x.key < y.key
6      CUT(H, x, y)
7      CASCADING-CUT(H, y)
8  if x.key < H.min.key
9      H.min = x

CUT(H, x, y)
1  remove x from the child list of y, decrementing y.degree
2  add x to the root list of H
3  x.p = NIL
4  x.mark = FALSE

CASCADING-CUT(H, y)
1  z = y.p
2  if z ≠ NIL
3      if y.mark == FALSE
```

```
4            y.mark = TRUE
5        else CUT(H, y, z)
6            CASCADING-CUT(H, z)
```

يعمل إجراء FIB-HEAP-DECREASE-KEY كما يلي. تضمن الأسطر 1-3 ألاً يكون المفتاح الجديد أكبر من المفتاح الحالي للعقدة x، ثم تسند المفتاح الجديد إلى x. إذا كان x جذرًا أو كان $x.key \geq y.key$ حيث y هو أبو x، فلا حاجة إلى تغييرات بنيوية لأن ترتيب الكومة وفق الأصغر لم يُخرق. تختبر الأسطر 4-5 هذا الشرط.

إذا جرى خرق ترتيب الكومة وفق الأصغر، فقد تحدث عدة تغييرات. نبدأ بقطع x في السطر 6. "يقطع" الإجراء CUT الرابط بين x وأبيها y جاعلاً، من x جذرًا.

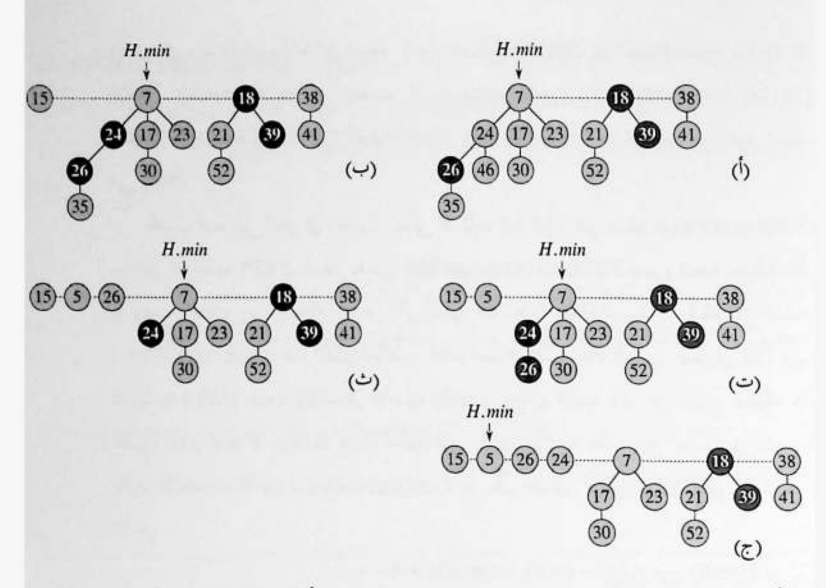
نستخدم الواصفات $mark$ للوصول إلى الحدود الزمنية المرغوبة، فهي تُسجِّل جزءًا صغيرًا من تاريخ كل عقدة. افترض أنَّ الأحداث التالية وقعت للعقدة x:

1. في وقت ما كانت x جذرًا،
2. ثم رُبطت x بعقدة أخرى (مكوِّنةً ابنًا لها)،
3. ثم جرى حذف ابنَيْن للعقدة x بالقطع.

بمجرد فقدان الابن الثاني، نفصل x عن أبيها جاعلين منها جذرًا جديدًا. تكون قيمة الواصفة $x.mark$ مساويةً TRUE إذا حدثت الخطوتان 1 و 2 وجرى قطع ابن واحد للعقدة x. فيقوم الإجراء CUT بمسح $x.mark$ في السطر 4 لأنه ينفِّذ الخطوة 1. (يمكننا الآن معرفة سبب مسح clear السطر 3 من FIB-HEAP-LINK للقيمة $y.mark$: فالعقدة y جرى ربطها بعقدة أخرى وبذلك تُنفَّذ الخطوة 2. في المرة التالية التي يجري فيها قطع أحد أبناء y ستصبح قيمة $y.mark$ هي TRUE.)

لم ننته بعد، لأن x قد تكون الابن الثاني المفصول عن الأب y منذ ربط y بعقدة أخرى. لذلك، يحاول السطر 7 من FIB-HEAP-DECREASE-KEY إجراء عملية ***قطع متتابع cascading-cut*** على y. إذا كانت y جذرًا فسيسبِّب الاختبار في السطر 2 من CASCADING-CUT عودة الإجراء فقط. إذا لم تكن y معلَّمة يقوم الإجراء بتعليمها في السطر 4، لأن ابنها الأول قد قُطع للتو، ثم يعود. لكن إذا كانت y معلَّمة، فهي قد فقدت للتو ابنها الثاني؛ فيجري قطع y في السطر 5 ويستدعي الإجراء CASCADING-CUT نفسه عوديًّا في السطر 6 على z أبي y. يستمر الإجراء CASCADING-CUT بالتنفيذ العودي إلى أعلى الشجرة حتى يصل إلى جذرٍ أو إلى عقدة غير معلَّمة.

عند انتهاء جميع عمليات القطع المتتابعة، يُنهي السطران 8–9 من FIB-HEAP-DECREASE-KEY العمل بتحديث $H.min$ إذا كان ذلك ضروريًّا. العقدة الوحيدة التي جرى تغيير مفتاحها كانت هي العقدة x



الشكل 5.19 استدعاءان للإجراء FIB-HEAP-DECREASE-KEY. (أ) كومة فيبوناتشي في البداية. (ب) جرى إنقاص مفتاح العقدة ذات المفتاح 46 إلى 15. تصبح هذه العقدة جذرًا، ويجري تعليم أبيها (العقدة ذات المفتاح 24) الذي لم يكن معلَّمًا. (ت)–(ج) يجري إنقاص مفتاح العقدة ذات المفتاح 35 إلى 5. في الجزء (ت) تصبح العقدة ذات المفتاح 5 جذرًا. يجري تعليم أبيها ذي المفتاح 26، فيحدث قطع متتابع. يجري فصل العقدة 26 عن أبيها وجعلها جذرًا غير معلَّم في (ث). يحدث قطع متتابع آخر لأن العقدة ذات المفتاح 24 هي أيضًا معلَّمة. فيجري فصلها عن أبيها وجعلها جذرًا غير معلَّم في الجزء (ج). يتوقف القطع المتتابع عند هذه النقطة لأن العقدة ذات المفتاح 7 هي جذر. (حتى إن لم تكن هذه العقدة جذرًا فسيتوقف القطع المتتابع لأنها غير معلَّمة.) تظهر نتيجة عملية FIB-HEAP-DECREASE-KEY في الجزء (ج)، مع المؤشر $H.min$ الذي يشير إلى العقدة الصغرى الجديدة.

التي جرى إنقاص قيمة مفتاحها. لذلك، فالعقدة الصغرى الجديدة هي إما العقدة الصغرى الأصلية وإما العقدة x.

يُظهر الشكل 5.19 تنفيذ استدعاءين للإجراء FIB-HEAP-DECREASE-KEY بدءًا من كومة فيبوناتشي الظاهرة في الشكل 5.19(أ). لا يتطلب الاستدعاء الأول المبيَّن في الشكل 5.19(ب) قطعًا متتابعًا. أما الاستدعاء الثاني المبيَّن في الشكل 5.19(ت)–(ج) فهو يستدعي عمليتي قطع متتابعتين.

سنبيِّن الآن أن التكلفة المخمَّدة للإجراء FIB-HEAP-DECREASE-KEY هي $O(1)$ فقط. نبدأ بتحديد التكلفة الفعلية. يأخذ الإجراء FIB-HEAP-DECREASE-KEY زمنًا $O(1)$، إضافةً إلى زمن إجراء القطع المتتابع. افترض أنَّ الإجراء FIB-HEAP-DECREASE-KEY استدعى الإجراء CASCADING-CUT عوديًّا c

مرة في طلب ما (الطلب المنشأ في السطر 7 من FIB-HEAP-DECREASE-KEY متبوعًا بـ $c-1$ طلب عودي للإجراء CASCADING-CUT). يتطلب كل استدعاء للإجراء CASCADING-CUT زمنًا $O(1)$ بدون الطلبات العودية. فتكون التكلفة الفعلية للإجراء FIB-HEAP-DECREASE-KEY مع جميع الطلبات العودية هي $O(c)$.

نحسب فيما يلي التغير في الكمون. لتكن H كومة فيبوناتشي قبل عملية FIB-HEAP-DECREASE-KEY مباشرةً. إنَّ طلب CUT في السطر 6 من FIB-HEAP-DECREASE-KEY يُنشئ شجرة جديدة جذرها العقدة x ويمسح clear خانة العلام في x (التي يمكن أن تكون FALSE سلفًا). يقطع كل استدعاء عودي لـ CASCADING-CUT، عدا الطلب الأخير، عقدة معلَّمة ويمسح خانة التعليم. فتحتوي كومة فيبوناتشي بعد التنفيذ $t(H)+c$ شجرة (الأشجار الأصلية $t(H)$ و $c-1$ شجرة ناتجة عن القطع المتتابع، والشجرة التي جذرها x) و $m(H)-c+2$ عقدة معلَّمة على الأكثر ($c-1$ عقدة أزيل تعليمها في القطع المتتابع وقد يكون الاستدعاء الأخير لـ CASCADING-CUT قد علَّم عقدة). فيكون بذلك التغير في الكمون هو على الأكثر

$$((t(H)+c)+2(m(H)-c+2))-(t(H)+2m(H))=4-c\ .$$

فتكون التكلفة المخمَّدة لـ FIB-HEAP-DECREASE-KEY هي على الأكثر

$$O(c)+4-c=O(1)\ ,$$

لأن بإمكاننا رفع قيمة وحدات الكمون لتطغى على الثابت المضمَّن في $O(c)$.

يمكن أن نَعْرف الآن سبب تعريف دالة الكمون ليتضمن حدًّا هو ضعف عدد العقد المعلَّمة. عندما يجري قطع عقدة معلَّمة y في قطع متتابع، يُمسَح تعليمها، فينقص الكمون بمقدار 2. تُدفع وحدة من الكمون للقطع ومَسْح بت التعليم، وتُصرف الوحدة الأخرى لزيادة وحدة الكمون بسبب تحول y إلى جذر.

حذف عقدة

يحذف شبه الرماز التالي عقدة من كومة فيبوناتشي ذات n عقدة بزمن مخمَّد $O(D(n))$. نفترض أنَّه لا يوجد حاليًّا مفتاح قيمته $-\infty$ في كومة فيبوناتشي.

```
FIB-HEAP-DELETE(H, x)
1  FIB-HEAP-DECREASE-KEY (H, x, −∞)
2  FIB-HEAP-EXTRACT-MIN(H)
```

يُصيِّر الإجراءُ FIB-HEAP-DELETE العقدةَ x العقدةَ الصغرى في كومة فيبوناتشي عن طريق إعطائها مفتاحًا فريدًا في صغره $-\infty$. ثم يَحذف الإجراءُ FIB-HEAP-EXTRACT-MIN العقدةَ x من كومة فيبوناتشي. إن

الزمن المخمَّد لـ FIB-HEAP-DELETE هو مجموع الزمن المخمَّد لـ FIB-HEAP-DECREASE-KEY وهو $O(1)$ والزمن المخمَّد لـ FIB-HEAP-EXTRACT-MIN وهو $O(D(n))$. وسنرى في المقطع 4.19 أنَّ $D(n) = O(\lg n)$، لذا فإن الزمن المخمَّد لـ FIB-HEAP-DELETE هو $O(\lg n)$.

تمارين

1-3.19

افترض أنَّ جذرًا x في كومة فيبوناتشي معلَّم. اشرح كيف أصبح x جذرًا معلَّمًا. بيِّن أنه ليس من المهم للتحليل كون x معلَّمًا، حتى لو لم يكن x جذرًا قد رُبط أولاً بعقدة أخرى ثم فَقَدَ ابنًا واحدًا.

2-3.19

برِّر الزمن المخمَّد $O(1)$ لإجراء FIB-HEAP-DECREASE-KEY باعتباره تكلفة وسطى للعمليات باستخدام تحليل مجمَّع aggregate analysis.

4.19 وضع حد للدرجة العظمى

لإثبات أنَّ الزمن المخمَّد لـ FIB-HEAP-EXTRACT-MIN و FIB-HEAP-DELETE هو $O(\lg n)$، يجب أن نُظهر أنَّ الحد الأعلى $D(n)$ لدرجة أية عقدة في كومة فيبوناتشي ذات n عقدة هو $O(\lg n)$. سنبين خصوصًا أنَّ $D(n) \le \lfloor \log_\phi n \rfloor$ حيث ϕ هي النسبة الذهبية المعرَّفة في المعادلة (24.3) كما يلي

$\phi = (1 + \sqrt{5})/2 = 1.61803 \ldots$.

يكون مبدأ التحليل كالتالي. لكل عقدة x في كومة فيبوناتشي، نُعرِّف size(x) على أنه عدد العقد في الشجرة الفرعية ذات الجذر x ومنها العقدة x نفسها. (لاحظ أنه ليس من الضروري أن تكون x ضمن لائحة الجذور بل قد تكون أية عقدة.) سنبين أنَّ size(x) يزداد أسيًّا بحسب $x.degree$. استحضر في ذهنك أنّه تجري المحافظة على $x.degree$ دومًا بحيث تساوي العدد الدقيق لدرجة x.

مبرهنة 1.19

لتكن x عقدة في كومة فيبوناتشي، ولنفترض أنَّ $x.degree = k$. ولتكن $y_1, y_2, \ldots, y_k$ أبناء x بترتيب ربطها بـ x من الأقدم إلى الأحدث. فيكون عند ذلك $y_1.degree \ge 0$ و $y_1.degree \ge i - 2$ لكل $i = 2, 3, \ldots, k$.

البرهان من البديهي أنَّ $y_1.degree \ge 0$.

في حالة $i \ge 2$، نلاحظ أنَّه عندما كانت y_i مرتبطة مع x، فإنَّ جميع $y_1, y_2, \ldots, y_{i-1}$ كانت أبناء

لـ x، ولا بد أنه كان لدينا $x.degree \geq i - 1$. ولمّا كانت العقدة y_i ترتبط مع العقدة x (باستخدام CONSOLIDATE) فقط إذا كانت $x.degree = y_i.degree$، فيجب أن يكون لدينا أيضًا $y_i.degree \geq i - 1$ في ذلك الوقت. ومنذ ذلك الوقت فقدت y_i على الأكثر ابنًا واحدًا، لأنها لو كانت فقدت ابنين لجرى قطعها من x (باستخدام CASCADING-CUT). وبذلك نستنتج أنَّ $y_i.degree \geq i - 2$. ■

وصلنا أخيرًا إلى الجزء التحليلي الذي يشرح الاسم "كومات فيبوناتشي". تذكَّر من المقطع 2.3 أنَّه في حالة $k = 0, 1, 2, \ldots$ يُعرَّف عدد فيبوناتشي من المرتبة k عوديًّا بالشكل:

$$F_k = \begin{cases} 0 & \text{if } k = 0 , \\ 1 & \text{if } k = 1 , \\ F_{k-1} + F_{k-2} & \text{if } k \geq 2 . \end{cases}$$

تقدِّم التوطئة الآتية طريقةً أخرى للتعبير عن F_k.

توطئة 2.19

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i .$$

لكل الأعداد الصحيحة $k \geq 0$

البرهان يجري البرهان بالتدريج على k. فعندما تكون $k = 0$

$$\begin{aligned} 1 + \sum_{i=0}^{0} F_i &= 1 + F_0 \\ &= 1 + 0 \\ &= F_2 . \end{aligned}$$

نفترض الآن الفرض التدريجي $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$، ويكون لدينا

$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} \\ &= F_k + \left(1 + \sum_{i=0}^{k-1} F_i \right) \\ &= 1 + \sum_{i=0}^{k} F_i . \end{aligned}$$

■

توطئة 3.19

يحقق عدد فيبوناتشي ذو الترتيب $(k + 2)$ المعادلة $F_{k+2} \geq \phi^k$، لكل الأعداد الصحيحة $k \geq 0$.

البرهان يجري البرهان بالتدريج على k. الحالتان الأساسيتان هما لـ $k = 0$ و $k = 1$. عندما تكون $k = 0$ يكون لدينا $F_2 = 1 = \phi^0$، وعندما يكون $k = 1$ يكون لدينا $F_3 = 2 > 1.619 > \phi^1$. الخطوة التدريجية لكل $k \geq 2$، ونفترض أنَّ $F_{i+2} > \phi^i$ لكل $i = 0, 1, \dots, k-1$. تذكَّر أنَّ ϕ هو الجذر الموجب للمعادلة (23.3)، $x^2 = x + 1$. فيكون لدينا

$$\begin{aligned} F_{k+2} &= F_{k+1} + F_k \\ &\geq \phi^{k-1} + \phi^{k-2} && \text{(بحسب فرضية التدرج)} \\ &= \phi^{k-2}(\phi + 1) \\ &= \phi^{k-2} \cdot \phi^2 && \text{(بحسب المعادلة (23.3))} \\ &= \phi^k . \end{aligned}$$

■

يكتمل التحليل بالتوطئة التالية مع نتيجتها.

توطئة 4.19

لتكن x عقدة ما في كومة فيبوناتشي، وليكن $k = x.degree$، فيكون $\text{size}(x) \geq F_{k+2} \geq \phi^k$ حيث $\phi = (1 + \sqrt{5})/2$.

البرهان نرمز بـ s_k للحجم الأصغر الممكن لأية عقدة من الدرجة k في أية كومة فيبوناتشي. من البديهي أنَّ $s_0 = 1$ و $s_1 = 2$. العدد s_k هو على الأكثر $\text{size}(x)$، ولأنَّ إضافة أبناء إلى عقدة ما لا يمكن أن يقلل من حجمها، فإنَّ قيمة s_k تزداد بانتظام بزيادة k. افترض أن عقدة ما z في كومة فيبوناتشي تحقق $z.degree = k$ و $\text{size}(z) = s_k$. ولمّا كان $s_k \leq \text{size}(x)$ فإننا نحسب حدًّا أدنى على $\text{size}(x)$ بحساب حد أدنى على s_k. وكما في التوطئة 1.19، لتكن $y_1, y_2, \dots, y_k$ هي أبناء z بترتيب ربطها بـ z. لحساب حد أصغر للقيمة s_k نحسب واحدًا من أجل z نفسها وواحدًا من أجل الابن الأول y_1 (والذي يحقق $\text{size}(y_1) \geq 1$) فيكون

$$\begin{aligned} \text{size}(x) &\geq s_k \\ &\geq 2 + \sum_{i=2}^{k} s_{y_i.degree} \\ &\geq 2 + \sum_{i=2}^{k} s_{i-2} , \end{aligned}$$

حيث ينتج السطر الأخير من تطبيق التوطئة 1.19 (وبذلك يكون $y_i.degree \geq i - 2$) ومن التزايد المنتظم s_k (وبذلك يكون $s_{y_i.degree} \geq s_{i-2}$).

نبين الآن بالتدريج على k أنَّ $s_k \geq F_{k+2}$ لكل الأعداد الصحيحة الموجبة k. الحالتان الأساسيتان لـ $k = 0$ و $k = 1$ بديهيتان. نفترض من أجل الخطوة التدريجية أنَّ $k \geq 2$ وأنَّ $s_i \geq F_{i+2}$ لكل $i = 0, 1, \ldots, k-1$، فيكون لدينا:

$$\begin{aligned} s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\ &\geq 2 + \sum_{i=2}^{k} F_i \\ &= 1 + \sum_{i=0}^{k} F_i \\ &= F_{k+2} && \text{((بحسب التوطئة (2.19))} \\ &\geq \phi^k && \text{((بحسب التوطئة (3.19))} \end{aligned}$$

وبذلك نكون قد أثبتنا أنَّ $\text{size}(x) \geq s_k \geq F_{k+2} \geq \phi^k$. ■

نتيجة 5.19

الدرجة العظمى $D(n)$ لعقدة ما في كومة فيبوناتشي ذات n عقدة هي $O(\lg n)$.

البرهان لتكن x عقدة ما في كومة فيبوناتشي ذات n عقدة، ولتكن $k = x.degree$. بحسب التوطئة 4.19 يكون $n \geq \text{size}(x) \geq \phi^k$. بأخذ اللغاريتم ذي الأساس ϕ يكون $k \leq \log_\phi n$. (في الحقيقة، $k \leq \lfloor \log_\phi n \rfloor$ لأنَّ k عدد صحيح.) فتكون بذلك الدرجة العظمى $D(n)$ لأي عقدة هي $O(\lg n)$. ■

تمارين

1-4.19

يؤكد الأستاذ Pinocchio أنَّ ارتفاع كومة فيبوناتشي ذات n عقدة هو $O(\lg n)$. بيِّن أن الأستاذ مخطئ من خلال عرض متتالية من عمليات كومات فيبوناتشي تُنشئ كومة فيبوناتشي مؤلفة من شجرة واحدة فقط على شكل سلسلة خطية من n عقدة، لأي عدد n صحيح موجب.

2-4.19

افترض أننا نعمم قاعدة القطع المتتابع بحيث نقطع عقدة x من أبيها مباشرةً بعد فقدها الابن ذا الترتيب k، حيث k عدد ثابت صحيح. (تَستخدم القاعدةُ في المقطع 3.19 القيمةَ $k = 2$.) ما هي قيم k التي تحقق $D(n) = O(\lg n)$؟

مسائل

1-19 تنجيز بديل للحذف

اقترح الأستاذ Pisano الشكل التالي للإجراء FIB-HEAP-DELETE، مدعيًا أنَّه يعمل بسرعة أكبر عندما لا تكون العقدة المحذوفة هي العقدة التي يشير إليها $H.min$.

```
PISANO-DELETE(H, x)
1  if x == H.min
2      FIB-HEAP-EXTRACT-MIN(H)
3  else y = x.p
4      if y ≠ NIL
5          CUT(H, x, y)
6          CASCADING-CUT(H, y)
7      add x's child list to the root list of H
8      remove x from the root list of H
```

أ. إنَّ ادعاء الأستاذ بأنَّ هذا الإجراء يُنفذ بسرعة أكبر يعتمد جزئيًا على افتراض أنَّ السطر 7 يمكن أن يُنفَّذ بزمن فعلي $O(1)$. ما هو الخطأ في هذا الافتراض؟

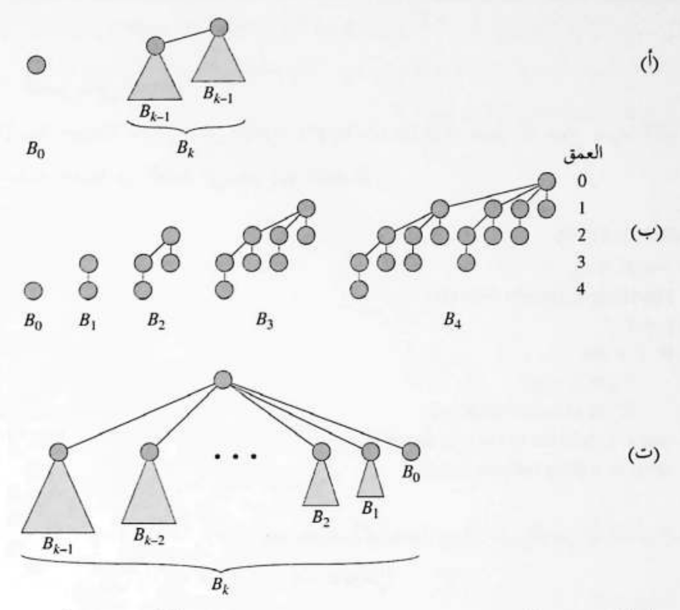
ب. أعطِ حدًّا أعلى جيدًا للزمن الفعلي للإجراء PISANO-DELETE عندما لا تكون x هي $H.min$. يجب أن يكون الحد الأعلى بدلالة $x.degree$ والعدد c الذي يمثِّل عدد استدعاءات الإجراء CASCADING-CUT.

ت. افترض أننا نستدعي الإجراء PISANO-DELETE(H,x)، ولتكن H' كومة فيبوناتشي الناتجة. بافتراض أنَّ العقدة x ليست جذرًا، ضع حدًّا لكمون H' بدلالة $x.degree$ و c و $t(H)$ و $m(H)$.

ث. استنتج أنَّ الزمن المخمَّد للإجراء PISANO-DELETE ليس أفضل بالمقارنة من زمن FIB-HEAP-DELETE، حتى عندما تكون $x \neq H.min$.

2-19 أشجار ثنائية الحد وكومات ثنائية الحد

الشجرة الثنائية الحد B_k هي شجرة مرتَّبة (انظر المقطع ب.2.5) تُعرَّف تعريفًا عوديًّا. وكما يَظهَر في الشكل 6.19(أ)، تتألف الشجرة الثنائية الحد B_0 من عقدة وحيدة. وتتألف الشجرة الثنائية الحد B_k من شجرتين ثنائيتي الحد B_{k-1} مرتبطتان إحداهما بالأخرى: جذر إحداهما هو الابن في أقصى اليسار لجذر الأُخرى. يبين الشكل 6.19(ب) الأشجار الثنائية الحد من B_0 إلى B_4.



الشكل 6.19 (أ) التعريف العودي للشجرة الثنائية الحد B_k. تُمثّل المثلثات أشجارًا فرعية ذات جذور. (**ب**) الأشجار الثنائية الحد من B_0 إلى B_4، تَظهر أعماق العقد في B_4. (**ت**) طريقة أخرى للنظر إلى الشجرة الثنائية الحد B_k.

أ. أثبت أنه في شجرة ثنائية الحد B_k

1. توجد 2^k عقدة.
2. ارتفاع الشجرة هو k.
3. هناك تمامًا $\binom{k}{i}$ عقدة في العمق i لكل $i = 0, 1, .., k$.
4. درجة الجذر هي k، وهي أكبر من درجة أية عقدة أخرى؛ إضافةً إلى أنه، كما يُظهر الشكل 6.19(ت)، إذا كانت أرقام أبناء الجذر من اليسار إلى اليمين: $k-1,\ k-2, \ldots, 0$، فإنَّ الابن i هو جذر للشجرة الفرعية B_i.

الكومة الثنائية الحد ***binomial heap H*** هي مجموعة من الأشجار الثنائية الحد تحقق الخصائص الآتية:

1. كل عقدة لها مفتاح (كما في كومات فيبوناتشي).
2. كل شجرة ثنائية الحد في H تخضع لخصائص الكومات المرتبة وفق الأصغر **min-heap property**.

3. في حالة أي عدد k صحيح غير سالب، يوجد على الأكثر شجرة ثنائية الحد في H جذرها من الدرجة k.

ب. افترض أنَّ كومة ثنائية الحد H فيها n عقدة. ناقش العلاقة بين الأشجار الثنائية الحد التي تحتويها H والتمثيل الاثناني لـ n. استنتج أنَّ H تتألف من $\lfloor \lg n \rfloor + 1$ شجرة ثنائية الحد.

افترض أننا نمثِّل كومة ثنائية الحد كالتالي. يُمثِّل أسلوب الابن الأيسر والأخ الأيمن المقدم في المقطع 4.10 كل شجرة ثنائية الحد ضمن كومة ثنائية الحد. تحتوي كل عقدة مفتاحها؛ ومؤشرًا إلى أبيها، وآخر على ابنها في أقصى اليسار، وثالث إلى أخيها الأيمن المباشر (هذه المؤشرات تكون NIL عند اللزوم)؛ ودرجتها (كما في كومات فيبوناتشي) عدد أولادها. تكوِّن الجذور لائحة جذور مترابطة وحيدة، مرتبة بحسب درجات الجذور (من الأدنى إلى الأعلى)، ويجري النفاذ إلى الكومة الثنائية الحد عن طريق مؤشر إلى العقدة الأولى في لائحة الجذور.

ت. أكمل وصف كيفية تمثيل كومة ثنائية الحد (سَمِّ الواصفات، وحدِّد متى تأخذ الواصفات القيمة NIL، وعرِّف كيفية تنظيم لائحة الجذور)، وأظهر كيفية تنجيز العمليات السبع نفسها على الكومات الثنائية الحد كما نجّزها هذا الفصل على كومات فيبوناتشي. يجب أن تُنفَّذَ كل عملية بزمن $O(\lg n)$ في أسوأ الحالات، حيث n هو عدد العقد في الكومة الثنائية الحد (أو في حالة عملية UNION، في الكومتين الثنائيتين اللتين يجري جمعهما). يجب أن تستغرق عملية MAKE-HEAP زمنًا ثابتًا.

ث. افترض أننا نريد تنجيز العمليات على الكومات القابلة للدمج فقط في كومات فيبوناتشي (أي لا نُنجّز عمليتي DECREASE-KEY أو DELETE). كيف يمكن أن تشبه الأشجار في كومة فيبوناتشي تلك الموجودة في كومة ثنائية الحد؟ بماذا تختلف عنها؟ بيِّن أنَّ الدرجة العظمى في كومة فيبوناتشي ذات n عقدة ستكون $\lfloor \lg n \rfloor$ على الأكثر.

ج. اخترَع المدرس McGee بنية معطيات جديدة تعتمد على كومات فيبوناتشي. كومة McGee لها بنية كومة فيبوناتشي نفسها وتدعم عمليات الكومات القابلة للدمج فقط. وطريقة تنجيز العمليات هي نفسها في كومات فيبوناتشي، إلا أنَّ الإدراج والاجتماع يُدَعِّمان لائحة الجذور في خطوتهما الأخيرة. ما هي أزمنة تنفيذ العمليات على كومات McGee في أسوأ الحالات ؟

3-19 المزيد من العمليات على كومات فيبوناتشي

نرغب بإغناء كومة فيبوناتشي H لتدعم عمليتين جديدتين دون تغيير زمن التنفيذ المخمَّد لأية عملية أخرى على كومات فيبوناتشي.

أ. العملية FIB-HEAP-CHANGE-KEY(H, x, k) تُغيِّر مفتاحَ العقدة x إلى القيمة k. أعطِ تنجيزًا فعالاً لهذا الإجراء، وحلِّل زمن التنفيذ المخمَّد لهذا التنجيز في الحالات التي تكون فيها k أكبر من المفتاح $x.key$ ، أو أصغر منه، أو مساوية له.

ب. أعطِ تنجيزًا فعالاً للإجراء FIB-HEAP-PRUNE(H, r)، الذي يحذف $q = \min(r, H.n)$ عقدة من H. يمكن أن تختار أي q عقدةً لتحذفها. حلِّل زمن التنفيذ المخمَّد لتنجيزك. (*تلميح*: قد تحتاج إلى تعديل بنية المعطيات ودالة الكمون.)

4-19 الكومات 2-3-4

قدَّم الفصل 18 الشجرة 2-3-4 التي يكون لكل عقدة داخلية فيها (ربما ماعدا الجذر) ابنان أو ثلاثة أو أربعة أبناء ولكل الأوراق العمق نفسه. في هذه المسألة سنقوم بتنجيز ***الكومات 2-3-4***، التي تدعم العمليات على الكومات القابلة للدمج.

تختلف الكومات 2-3-4 عن الأشجار 2-3-4 في المناحي التالية: في الكومات 2-3-4، الأوراق فقط هي التي تخزِّن المفاتيح، وكل ورقة x تخزن مفتاحًا واحدًا فقط في الواصفة $x.key$. يمكن أن تظهر المفاتيح في الأوراق بأي ترتيب. تحتوي كل عقدة داخلية x قيمة $x.small$ تساوي أصغر مفتاح مخزن في أي ورقة في شجرة فرعية جذرها x. يحتوي الجذر r واصفةً $r.height$ هو ارتفاع الشجرة. أخيرًا، الكومات 2-3-4 مُعَدَّة لإبقائها في الذاكرة الرئيسية بحيث لا نحتاج إلى القراءة من القرص أو الكتابة عليه.

نجِّز العمليات التالية على الكومات 2-3-4. في الأجزاء (أ)–(ج)، أي عملية يجب أن تنفذ بزمن $O(\lg n)$ على كومات 2-3-4 ذات n عنصرًا. عملية UNION في الجزء (و) يجب أن تنفذ بزمن $O(\lg n)$، حيث n هو عدد العناصر في كومتَي الدخل.

أ. العملية MINIMUM التي تعيد مؤشرًا إلى الورقة التي تحتوي المفتاح الأصغري.

ب. العملية DECREASE-KEY التي تنقص مفتاح ورقة معينة x إلى قيمة محددة $k \leq x.key$.

ت. العملية INSERT التي تدرج ورقة x مفتاحها k.

ث. العملية DELETE التي تحذف ورقة معينة x.

ج. العملية EXTRACT-MIN التي تنزع الورقة ذات المفتاح الأصغري.

ح. العملية UNION التي توحِّد كومتين من نوع 2-3-4، وتعيد كومة واحدة 2-3-4، وتدمِّر كومتَي الدخل.

ملاحظات الفصل

أَدْخَلَ Fredman و Tarjan [114] كومات فيبوناتشي. تصف هذه المقالة أيضًا تطبيق كومات فيبوناتشي على مسائل أقصر المسارات من منبع وحيد، وأقصر المسارات من أية عقدة إلى أية عقدة، والمزاوجة الثنائية الجزء المثقلة، ومسألة شجرة المسح الصغرى.

بعد ذلك، طوَّر Driscoll و Gabow و Shraiman و Tarjan [97] "الكومات المرخاة" "relaxed heaps" كبديل عن كومات فيبوناتشي. وحدَّدوا صنفان من الكومات المرخاة. يعطي أحدهما حدود كومات فيبوناتشي الزمنية المخمَّدة نفسها. ويسمح الآخر بتنفيذ DECREASE-KEY بزمن $O(1)$ في أسوأ الحالات (غير مخمَّد)، وبتنفيذ EXTRACT-MIN و DELETE بزمن $O(\lg n)$ في أسوأ الحالات. كذلك فإنَّ للكومات المرخاة بعض الميزات على كومات فيبوناتشي في الخوارزميات المتوازية.

ارجع أيضًا إلى ملاحظات الفصل 6 ففيها معلوماتٌ عن بنى معطيات أخرى تدعم عمليات DECREASE-KEY سريعة عندما تكون متتالية القيم التي تعيدها استدعاءات EXTRACT-MIN متزايدة بانتظام عبر الزمن، وتكون المعطيات أعدادًا صحيحة في مجال محدد.

20 أشجار Van Emde Boas

شاهدنا في فصول سابقة بنًى معطياتٍ تدعم عمليات الأرتال ذات الأولوية: الكومات الثنائية في الفصل 6، والأشجار الحمراء–السوداء في الفصل 13،[1] وكومات فيبوناتشي في الفصل 19. ورأينا أنَّ في كلِّ بنيةٍ من بنى المعطيات هذه، توجد عمليةٌ هامةٌ واحدةٌ على الأقل تستغرق زمنًا $O(\lg n)$، إما في أسوأ الحالات وإما في الحالة المخمدة. والواقع أنه لما كانت جميع بنى المعطيات هذه تعتمد في قراراتها على مقارنة المفاتيح، فإن الحد الأدنى للفرز $\Omega(n \lg n)$، الوارد في المقطع 1.8، يعني أنَّ عمليةً واحدة على الأقل يجب أن تستغرق زمنًا $\Omega(\lg n)$. لماذا؟ إذا استطعنا إجراء عمليتَي INSERT و EXTRACT-MIN بزمن $o(\lg n)$، يمكننا عندها فرز n مفتاحًا بزمن $o(n \lg n)$ بإجراء n عملية INSERT أولاً، ثم n عملية EXTRACT-MIN.

غير أننا شاهدنا في الفصل 8، أن بإمكاننا أحيانًا استغلال معلومات إضافية عن المفاتيح لإجراء الفرز بزمن $o(n \lg n)$. ويمكننا، بصورة خاصة في الفرز بالعد، فرزَ n مفتاحًا، كلٌّ منها هو عددٌ صحيح يقع ضمن المجال من 0 إلى k، بزمن $\Theta(n + k)$، والذي هو $\Theta(n)$ في حال كانت $k = O(n)$.

ولما كان باستطاعتنا الالتفاف حول الحد الأدنى للفرز $\Omega(n \lg n)$ عندما تكون المفاتيح أعدادًا صحيحة ضمن مجال محدود، يمكنك أن تتساءل: هل نستطيع، بأسلوب مشابه، إجراء كلٍّ من عمليات الرتل ذي الأولوية بزمن $o(\lg n)$؟ سنرى في هذا الفصل أن ذلك ممكن: إذ إنَّ أشجار van Emde Boas تدعم عمليات الرتل ذي الأولوية، وبعض العمليات الأخرى بزمن $O(\lg \lg n)$ في أسوأ الحالات. الفكرة هنا هي أن المفاتيح يجب أن تكون أعدادًا صحيحة ضمن المجال الممتد من 0 إلى $n - 1$، دون السماح بتكرار أيٍّ منها.

تدعمُ أشجار van Emde Boas، على وجه الخصوص، كلاً من العمليات الآتية على المجموعات الديناميكية المسرودة في الصفحة 230 وهي: SEARCH و INSERT و DELETE و MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR، وذلك بزمن $O(\lg \lg n)$. ولن نناقش، في هذا الفصل، المعطيات التابعة،

---

[1] لا يناقش الفصل 13 صراحة كيفية تنجيز EXTRACT-MIN و DECREASE-KEY، ولكن بإمكاننا بناء هذه العمليات بسهولة في أية بنية معطيات تدعم العمليات MINIMUM و DELETE و INSERT.

بل سنركز فقط على تخزين المفاتيح، وذلك لأننا سنركز على المفاتيح ولن نسمح بتخزين مفاتيح متكررة، فبدلاً من وصف عملية SEARCH، سننجز العملية الأبسط $\text{MEMBER}(S, x)$، التي تعيد قيمةً منطقيةً تدل على وجود القيمة x حاليًّا في المجموعة الديناميكية S أم لا.

استخدمنا حتى الآن الموسط n لغرضَيْن متمايزين: أولهما عدد العناصر في المجموعة الديناميكية، وثانيهما مجال القيم المحتملة. ولتجنب أي التباس آخر، سنستخدم من الآن فصاعدًا n للدلالة على عدد العناصر الموجودة حاليًّا في المجموعة، و u للدلالة على مجال القيم المحتملة، وبذلك، فإن كلَّ عملية من عمليات أشجار van Emde Boas تنفَّذ بزمن $O(\lg \lg u)$. نسمي المجموعة $\{0, 1, 2, \dots, u-1\}$ ***عالَم القيم universe of values*** التي يمكن تخزينها، و u ***حجم العالَم universe size***. نفترض في هذا الفصل أن u هو من مضاعفات العدد 2، أي $u = 2^k$ حيث k عدد صحيح أكبر أو يساوي الواحد.

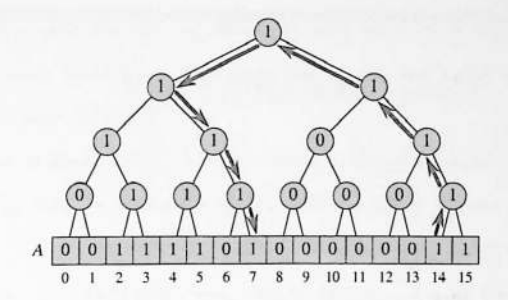
يبدأ المقطع 1.20 بفحص بعض المنهجيات البسيطة التي ستجعلنا نسير في الاتجاه الصحيح. ثم نحسِّن هذه المنهجيات في المقطع 2.20، بتقديم بنى proto van Emde Boas، التي هي بنًى عودية ولكنها لا تحقق غرضنا المتعلق بعمليات ذات زمن $O(\lg \lg u)$. وفي المقطع 3.20 نعدِّل بنى proto-van Emde Boas لإنشاء أشجار van Emde Boas، ونبيِّن كيفية تنجيز كلِّ عملية بزمن $O(\lg \lg u)$.

1.20 منهجيات مبدئية

سنفحص، في هذا المقطع، منهجياتٍ متعددةً لتخزين مجموعةٍ ديناميكية. ومع أن أيًّا من هذه المنهجيات لن تحقِّق في الزمن المرغوب $O(\lg \lg u)$، فإننا سنحصل على أفكار تساعدنا على فهم أشجار van Emde Boas عندما تمرّ بنا لاحقًا في هذا الفصل.

العنونة المباشرة

توفر العنونة المباشرة direct addressing – كما رأينا في المقطع 1-11 – أبسط منهجية لتخزين مجموعة ديناميكية. ولما كان اهتمامنا في هذا الفصل محصورًا في تخزين المفاتيح فقط، فيمكننا تبسيط منهجية العنونة المباشرة لتخزين المجموعة الديناميكية، وذلك باعتبارها شعاع بتات bit vector [انظر المناقشة في التمرين 2-1.11]. فلتخزين مجموعةٍ ديناميكيةٍ من قيم العالَم $\{0, 1, 2, \dots, u-1\}$، نَحفظُ صفيفة $A[0 \,..\, u-1]$ من u بتًّا. حيث يأخذ العنصر $A[x]$ القيمةَ 1 إذا كانت القيمة x ضمن المجموعة الديناميكية، والقيمة 0 في الحالة الأخرى. ومع أن بإمكاننا إجراءَ كلٍّ من العمليات INSERT و DELETE و MEMBER بزمن $O(1)$ باستخدام شعاع بتات، فإن كلاً من العمليات المتبقية – MINIMUM و MAXIMUM و SUCCESSOR و PREDELESSOR – تستغرق زمنًا $\Theta(u)$ في أسوأ الحالات، وذلك لأننا ربما نضطر إلى مسح scan $\Theta(u)$



الشكل 1.20 شجرة ثنائية من بتات مُراكَبَة فوق شعاع بتات يمثل المجموعة {2, 3, 4, 5, 7, 14, 15} في حالة $u = 16$. تتضمن كل عقدة داخلية 1 إذا وفقط إذا تضمنت ورقةٌ ما في أشجارها الفرعية القيمةَ 1. وتبيِّن الأسهمُ المسارَ المتَّبع لتحديد العنصر السابق للقيمة 14 في المجموعة.

عنصرًا.[2] فعلى سبيل المثال، إذا تضمنت مجموعة ما القيمتين 0 و $u - 1$ فقط، فقد نضطر – عند العثور على العنصر التالي للعنصر 0 – إلى مَسْح العناصر من 1 إلى $u - 2$ قبل العثور على 1 في $A[u - 1]$.

مراكبة بنية شجرة ثنائية في الأعلى

يمكننا اختصار عمليات المسح الطويلة لشعاع البتات بمراكبة شجرة ثنائية من البتات أعلى منه. يبين الشكل 1.20 مثالاً على ذلك. تكوِّن عناصرُ شعاع البتات أوراقَ الشجرة الثنائية، وتتضمن كلُّ عقدةٍ داخليةٍ القيمةَ 1 إذا وفقط إذا تضمنت أيةُ ورقةٍ من شجرتها الفرعيةِ القيمةَ 1. بعبارة أخرى، فإن البت المخزّن في عقدة داخلية هو نتيجة إجراء عملية "أو المنطقية" على ابْنَيْها.

تَستخدم العملياتُ – التي استغرقت باستخدام شعاع بتات بسيط زمنًا $\Theta(u)$ في أسوأ الحالات – البنيةَ الشجريةَ الآن:

- للعثور على القيمة الدنيا في المجموعة، ابدأ من الجذر واتجه نزولاً نحو الأوراق، بحيث تأخذ دومًا العقدة في أقصى اليسار التي تتضمن القيمة 1.
- للعثور على القيمة العظمى في المجموعة، ابدأ من الجذر واتجه نزولاً نحو الأوراق، بحيث تأخذ دومًا العقدة في أقصى اليمين التي تتضمن القيمة 1.

---

[2] نفترض في هذا الفصل أن MINIMUM و MAXIMUM تعيدان NIL إذا كانت المجموعة الديناميكية خالية، وأن SUCCESSOR و PREDECESSOR تعيدان NIL إذا لم يكن للعنصر المعطى عنصر لاحق أو سابق على التتالي.

- للعثور على العنصر التالي successor لـ x، ابدأ من الورقة التي دليلها x، واتجه صعودًا نحو الجذر حتى تَدخُلَ في عقدة من اليسار ويكون لهذه العقدة ابنًا أيمن z قيمته 1. ثم اتجه نزولاً عبر العقدة z، بحيث تأخذ دومًا العقدة في أقصى اليسار التي تتضمن القيمة 1 (أي، اعثر على القيمة الدنيا في الشجرة الفرعية التي جذرها الابن الأيمن z).
- للعثور على العنصر السابق predecessor لـ x، ابدأ من الورقة التي دليلها x، واتجه صعودًا نحو الجذر حتى تَدخُلَ في عقدة من اليمين ويكون لهذه العقدة ابنًا أيسر z قيمته 1. ثم اتجه نزولاً عبر العقدة z، بحيث تأخذ دومًا العقدة في أقصى اليسار التي تتضمن القيمة 1 (أي، اعثر على القيمة العظمى للشجرة الفرعية التي جذرها الابن الأيسر z).

يبين الشكل 1.20 المسار المسلوك لإيجاد العنصر السابق 7 للقيمة 14.

نوسِّع كذلك عمليتَي INSERT و DELETE توسيعًا ملائمًا. فعند إدراج قيمةٍ، نخزن القيمة 1 في كل عقدة موجودة على المسار البسيط الممتد من الورقة الموافقة وحتى الجذر. وعند حذف قيمة، نسير انطلاقًا من الورقة الموافقة صعودًا باتجاه الجذر، بحيث نعيد حساب البت في كل عقدة داخلية من المسار على أنه نتيجة تطبيق "أو المنطقية" على ابْنَيْه.

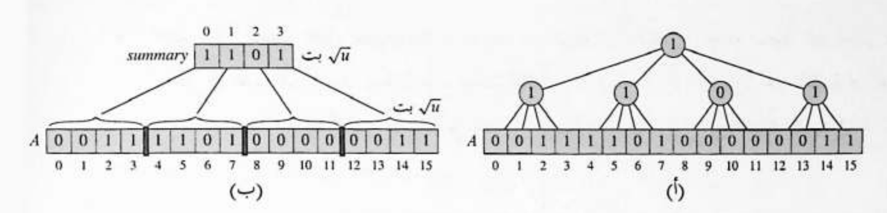
ولما كان ارتفاع الشجرة هو $\lg u$، وكانت كلُّ عمليةٍ من العمليات السابقة تتطلب، على الأكثر، عبورًا واحدًا للشجرة باتجاه الأعلى، وعلى الأكثر، عبورًا آخر باتجاه الأسفل، فإن كل عملية تستغرق زمنًا $O(\lg u)$ في أسوأ الحالات.

هذه المنهجية أفضل قليلاً فقط من استخدام شجرة حمراء-سوداء. حيث مازال بإمكاننا إنجاز عملية MEMBER بزمن $O(1)$، في حين سيأخذ البحث في شجرة حمراء-سوداء زمنًا $O(\lg n)$. وهكذا نجد ثانية أنه إذا كان عدد العناصر n المخزنة أصغر بكثير من حجم العالَم u، فستكون الشجرة الحمراء-السوداء أسرع في جميع العمليات الأخرى.

مراكبة شجرة ذات ارتفاع ثابت

ما الذي يحدث إذا راكبنا شجرة ذات درجة أعلى؟ لنفترض أن حجم الفضاء هو $u = 2^{2k}$، حيث k عدد صحيح، فيكون $\sqrt{u}$ عددًا صحيحًا. فبدلاً من أن نراكب شجرةً ثنائية فوق شعاع البتات، نراكب شجرةً درجتها $\sqrt{u}$. يبين الشكل 2.20(أ) شجرةً مماثلةً لشعاع البتات نفسه الذي في الشكل 1.20. إن ارتفاع الشجرة الناتجة هو 2 دومًا.

كما في السابق، تُخزِّن كلُّ عقدةٍ داخليةٍ نتيجةَ تطبيق "أو المنطقية" على البتات ضمن شجرتها الفرعية، بحيث تلخِّص العقدُ الـ $\sqrt{u}$ الداخلية، التي عمقها 1، كلَّ مجموعةٍ من $\sqrt{u}$ قيمة. وكما يبين الشكل 2.20(ب)، يمكننا اعتبار هذه العقد صفيفة $summary[0..\sqrt{u}-1]$، حيث تتضمن



الشكل 2.20 (أ) شجرة درجتها $\sqrt{u}$ مراكبة فوق شعاع البتات الموجود في الشكل 1.20. تُخزِّن كلُّ عقدةٍ داخلية قيمةَ "أو المنطقية" للبتات في الشجرة الفرعية. **(ب)** منظر للبنية نفسها عندما تعامَل العقدُ الداخلية على العمق 1 باعتبارها صفيفة $summary[0..\sqrt{u}-1]$، حيث $summary[i]$ هي قيمة "أو المنطقية" للصفيفة الجزئية $A[i\sqrt{u}..(i+1)\sqrt{u}-1]$.

$summary[i]$ القيمةَ 1 إذا وفقط إذا تضمنت الصفيفةُ الجزئيةُ $A[i\sqrt{u}..(i+1)\sqrt{u}-1]$ القيمةَ 1. نسمي هذه الصفيفة الجزئية من A ذات الـ $\sqrt{u}$ بتًّا ***العنقودَ cluster*** ذا الترتيب i. في حالة قيمة معطاة x، يَظهر البت $A[x]$ في العنقود رقم $\lfloor x/\sqrt{u} \rfloor$. وهكذا أصبح الإجراء INSERT يستغرق الآن زمنًا $O(1)$: ولإدراج x، ضع القيمة 1 في كلٍّ من $A[x]$ و $summary[\lfloor x/\sqrt{u} \rfloor]$. يمكننا استخدام صفيفة $summary$ لإنجاز كلٍّ من العمليات: MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR و DELETE بزمنٍ $O(\sqrt{u})$:

- للعثور على القيمة الدنيا (العظمى)، ابحث عن العنصر الذي يتضمن 1 ويقع في أقصى يسار (يمين) $summary$، وليكن $summary[i]$، ثم ابحث خطيًّا – ضمن العنقود ذي الترتيب i – عن القيمة 1 الموجودة في أقصى اليسار (اليمين).
- للعثور على العنصر التالي (السابق) للعنصر x، ابحث أولاً باتجاه اليمين (اليسار) ضمن العنقود. فإذا وجدتَ القيمةَ 1، فيكون هذا الموقع هو النتيجة. وإلا، فاجعل $i = \lfloor x/\sqrt{u} \rfloor$، وابحث باتجاه اليمين (اليسار) ضمن صفيفة $summary$ ابتداءً من الدليل i. إن أول موقعٍ يتضمن القيمةَ 1 يعطينا دليل عنقود. ابحث ضمن هذا العنقود عن أول 1 في أقصى اليسار (اليمين). هذا الموقع يحوي العنصر التالي (السابق).
- لحذف القيمة x، اجعل $i = \lfloor x/\sqrt{u} \rfloor$. ضع القيمة 0 في $A[x]$، ثم ضع في $summary[i]$ قيمة ناتج "أو المنطقية" للبتات في العنقود ذي الترتيب i.

في كلٍّ من العمليات السابقة، نبحث، على الأكثر، في عنقودين من $\sqrt{u}$ بتًّا، إضافة إلى الصفيفة $summary$، وهكذا فإن كل عملية تستغرق زمنًا $O(\sqrt{u})$.

يبدو، للوهلة الأولى، وكأننا أجرينا تعديلاً سلبيًّا. أعطتنا مراكبة شجرة ثنائية عملياتٍ بزمن $O(\lg u)$، والتي هي أسرع بالمقاربة من زمن $O(\sqrt{u})$. ولكن، سيتضح أن استخدام شجرة من درجة $\sqrt{u}$ هي فكرة أساسية لأشجار van Emde Boas. سنتابع هذا المسار في المقطع التالي.

تمارين

1-1.20

عدّل بنى المعطيات في هذا المقطع لتدعم المفاتيح المتكررة.

2-1.20

عدّل بنى المعطيات في هذا المقطع لتدعم المفاتيح التي لها معطيات تابعة مرفقة.

3-1.20

لاحظ أن الطريقة التي نَجِدُ فيها العنصرَ التالي والسابق لقيمةٍ ما x - باستخدام البنى في هذا المقطع - لا تعتمد على كون x موجودة في المجموعة وقتئذٍ. بيّن كيف يمكنك العثور على العنصر التالي لـ x في شجرة بحث ثنائية عندما تكون x غير مخزنة في الشجرة.

4-1.20

افترض أنه بدلاً من مراكبة شجرةٍ درجتُها $\sqrt{u}$، راكبنا شجرةً درجتُها $u^{1/k}$، حيث k ثابت أكبر من الواحد. ماذا سيكون ارتفاع هذه الشجرة؟ وكم ستستغرق كل عملية من العمليات؟

2.20 بنية عودية

نعدّل في هذا المقطع فكرة مراكبة شجرة درجتها $\sqrt{u}$ فوق شعاع بتات. فقد استخدمنا في المقطع السابق بنيةً مختصرةً حجمُها $\sqrt{u}$، وكلُّ عنصرٍ فيها يشير إلى بنية أخرى حجمُها $\sqrt{u}$. أما حاليًّا، فنجعل البنية عودية، مقلِّصين حجمَ الفضاء إلى جذره، في كل مستوىً من العودية. وابتداءً من فضاءٍ حجمُه u، نجعل البنى تتضمن $\sqrt{u} = u^{1/2}$ عنصرًا، والتي تتضمن بدورها بنىً من $u^{1/4}$ عنصرًا، والتي تتضمن بدورها بنىً من $u^{1/8}$ عنصرًا، وهكذا نزولاً حتى الوصول إلى حجم أساسي هو 2.

نفترض للتبسيط، في هذا المقطع، أن $u = 2^{2^k}$ حيث k عدد صحيح، وبحيث تكون $u, u^{1/2}, u^{1/4}, \dots$، أعدادًا صحيحة. قد يكون هذا القيد قاسيًا عمليًّا، ويسمح أن تكون قيم u ضمن المتتالية ... ,2, 4, 16, 256, 65536 فقط. سنرى في المقطع التالي كيف نخفِّف هذا القيد، ونفترض أن $u = 2^k$ فقط حيث k عدد صحيح. ولما كانت البنية التي نفحصها في هذا المقطع هي بنيةٌ تمهيدية فقط لشجرة van Emde Boas الفعلية، فإننا نتساهل بخصوص هذا القيد للمساعدة على فهم المسألة.

وحيث إن هدفنا هو تحقيق أزمنةِ تنفيذِ العمليات من رتبة $O(\lg\lg u)$، فلنفكر في كيفية الحصول على أزمنةِ تنفيذٍ كهذه. كنا قد رأينا في نهاية المقطع 3.4 أنه بتغيير المتحولات يمكننا إثبات أن حل المعادلة التكرارية:

$$T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \lg n \qquad (1.20)$$

هو $T(n) = O(\lg n \lg\lg n)$. لنأخذ معادلة تكرارية مماثلة، ولكنها أبسط:

$$T(u) = T(\sqrt{u}) + O(1)\ . \qquad (2.20)$$

فإذا استخدمنا التقنية نفسها؛ أي تغيير المتحولات، أمكننا إثبات أن حل المعادلة التكرارية (2.20) هو $T(u) = O(\lg\lg u)$. ليكن $m = \lg u$، وبحيث أن $u = 2^m$ ويكون لدينا:

$$T(2^m) = T(2^{m/2}) + O(1)\ .$$

وبتغيير الاسم $S(m) = T(2^m)$ نحصل على المعادلة التكرارية الجديدة.

$$S(m) = S(m/2) + O(1)\ .$$

وباستخدام الحالة 2 من الطريقة العامة master، يكون حل هذه المعادلة التكرارية هو $S(m) = O(\lg m)$. نعيد تغيير الاسم من $S(m)$ إلى $T(u)$ فينتج $T(u) = T(2^m) = S(m) = O(\lg m) = O(\lg\lg u)$.

ستوجّه المعادلةُ التكراريةُ 2.20 بحثَنا عن بنية معطيات. لذا سنصمِّم بنيةَ معطيات عودية تتقلص بمقدار $\sqrt{u}$ في كلِّ مستوىً من عوديتها. عندما تَعْبُرُ عمليةٌ بنيةَ المعطيات هذه، فإنها تستغرق زمنًا ثابتًا في كلِّ مستوًى قبل أن تنتقل عوديًّا إلى المستوى الأدنى. حينئذ ستحدِّد المعادلةُ التكرارية (2.20) زمنَ تنفيذ العملية.

فيما يلي طريقةٌ أخرى للتفكير بكيفية الحصول على الحد $\lg\lg u$ عند حل المعادلة التكرارية (2.20). عندما ننظر إلى حجم الفضاء في كل مستوىً من بنية المعطيات العودية، نجد المتتالية $u, u^{1/2}, u^{1/4}, u^{1/8}, \ldots$. فإذا أخذنا بالحسبان كمية البتات التي نحتاج إليها لتخزين حجم الفضاء في كل مستوى، فإننا نحتاج إلى $\lg u$ في المستوى الأعلى، ويحتاج كل مستوى إلى نصف بتات المستوى السابق. وبوجهٍ عام، إذا بدأنا بـ b بتًّا وبنصف عدد البتات في كل مستوى، سنحصل بعد $\lg b$ مستوىً على بتٍّ واحدٍ فقط. ولما كانت $b = \lg u$، فسيكون لدينا فضاءٌ حجمه 2، بعد $\lg\lg u$ مستوىً.

وبالعودة إلى بنى المعطيات في الشكل 2.20، نجد أن قيمةً معطاة x تقع في العنقود رقم $\lfloor x/\sqrt{u}\rfloor$. فإذا كنا ننظر إلى x على أنه عددٌ صحيحٌ ممثلٌ ثنائيًّا بـ $\lg u$ بتًّا، فإن رقم ذلك العنقود، $\lfloor x/\sqrt{u}\rfloor$، يعطَى بالبتات $(\lg u)/2$ الأكثر أهمية في x. ويظهر x، ضمن هذا العنقود، في الموقع $x \bmod \sqrt{u}$، والذي يعطَى بالبتات $(\lg u)/2$ الأقل أهمية في x. ولما كنا بحاجةٍ إلى الفهرسة بهذه الطريقة، فإننا نعرِّف بعض الدوال التي تساعدنا على إجراء ذلك:

$$\text{high}(x) = \lfloor x/\sqrt{u}\rfloor\ ,$$

$$\text{low}(x) = x \bmod \sqrt{u} \,,$$
$$\text{index}(x,y) = x/\sqrt{u} + y \,.$$

تعطينا الدالة $\text{high}(x)$ البتات $(\lg u)/2$ الأكثر أهمية في x، مولِّدةً رقمَ عنقودِ x. وتعطينا الدالة $\text{low}(x)$ البتات $(\lg u)/2$ الأقل أهمية في x، وتعطي موقعَ x ضمن عنقوده. أما الدالةُ $\text{index}(x,y)$، فتبني رقمَ عنصرٍ ابتداءً من x و y، بحيث تعامِل x على أنه البتات $(\lg u)/2$ الأكثر أهمية في رقم العنصر، و تعامِل y على أنه البتات $(\lg u)/2$ الأقل أهمية. وتصبح لدينا المساواة $x = \text{index}(\text{high}(x), \text{low}(x))$. ستكون قيمة u المستخدَمة في كلٍّ من هذه الدوال هي دومًا حجم عالَم بنية المعطيات التي نستدعي ضمنها الدالة، الذي سيتغير أثناء النُّزول في البنية العودية.

1.2.20 بنى proto van Emde Boas

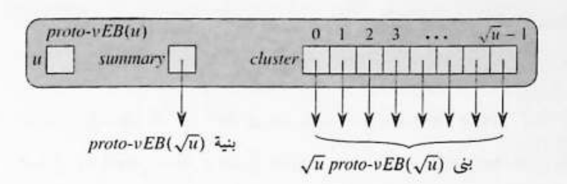
نصمِّم، انطلاقًا من المعادلة التكرارية (2.20)، بنية معطيات عودية تدعم العمليات. ومع أن هذه البنية لن تحقِّق هدفنا في الوصول إلى زمن $O(\lg \lg u)$ لبعض العمليات، فإنها ستَخْدِمُ باعتبارها أساسًا لبنية شجرة van Emde Boas التي سنراها في المقطع 3.20.

نعرّف ضمن العالَم $\{0,1,2,\ldots,u-1\}$ بنيةَ ***proto van Emde Boas***، أو بنية ***proto-vEB*** التي سنرمز لها بـ $proto\text{-}vEB(u)$، عوديًّا كما يلي: تتضمن كلُّ بنية $proto\text{-}vEB(u)$ واصفةً u تحدِّد حجم عالَمها. وهي تتضمن، إضافة إلى ذلك، ما يلي:

- إذا كان $u = 2$، عندها يكون هو حجم الأساس، وتتضمن البنيةُ صفيفةً $A[0..1]$ مؤلفةً من بتَّيْن.
- وإلا، يكون $u = 2^{2^k}$ حيث $k \geq 1$ عدد صحيح، وبذلك يكون $u \geq 4$. تتضمن بنية المعطيات $proto\text{-}vEB(u)$، إضافةً إلى حجم العالَم u، الواصفات التالية المبيَّنة في الشكل 3.20:
  - مؤشرًا إلى بنية $proto\text{-}vEB(\sqrt{u})$ اسمه $summary$، و
  - صفيفةً $cluster[0..\sqrt{u}-1]$ من $\sqrt{u}$ مؤشرًا، يشير كلٌّ منها إلى بنية $proto\text{-}vEB(\sqrt{u})$.

يخزَّن العنصر x، حيث $0 \leq x < u$، عوديًّا في العنقود رقم $\text{high}(x)$ على أنه العنصر $\text{low}(x)$ في ذلك العنقود.

في البنية الثنائية المستوى التي عرضناها في المقطع السابق، تخزِّنُ كلُّ عقدةٍ صفيفةَ ملخص summary حجمها $\sqrt{u}$، بحيث يتضمن كل عنصر فيها بتًّا واحدًا. ويمكننا، انطلاقًا من دليلِ كلِّ عنصرٍ، حسابُ دليلِ البدايةِ للصفيفة الجزئية التي حجمها $\sqrt{u}$ والتي يلخصها البت. نستخدم في بنية proto-vEB، مؤشراتٍ صريحةً بدلاً من حسابات الأدلة. تتضمن الصفيفةُ $summary$ بتات الملخص التي تخزَّن عوديًّا في بنية proto-vEB، أما الصفيفة $cluster$ فتتضمن $\sqrt{u}$ مؤشرًا.

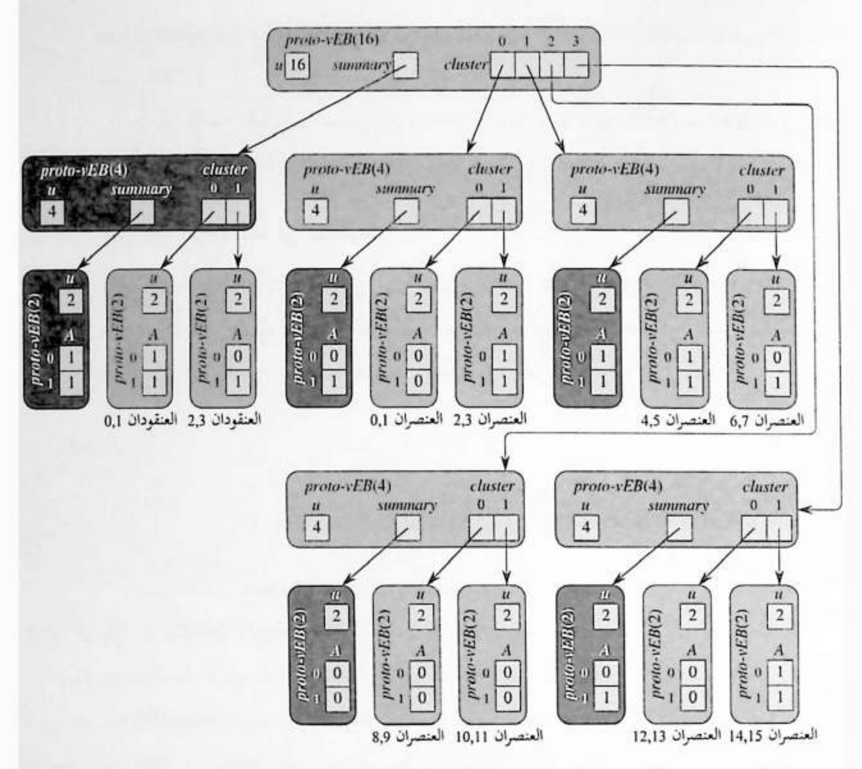


الشكل 3.20 المعلومات في بنية $proto\text{-}vEB(u)$ عندما تكون $u \geq 4$. تتضمن البنية: حجمَ الفضاء u، ومؤشرًا $summary$ على بنية $proto\text{-}vEB(\sqrt{u})$، وصفيفةً $cluster[0..\sqrt{u}-1]$ من $\sqrt{u}$ مؤشرًا على بنَى $proto\text{-}vEB(\sqrt{u})$.

يبين الشكل 4.20 بنية $proto\text{-}vEB(16)$ موسَّعة تمامًا تمثِّل المجموعة $\{2,3,4,5,7,14,15\}$. فإذا كانت القيمةُ i موجودةً في بنية proto-vEB التي يشير إليها $summary$، فإن العنقودَ ذا الترتيب i يتضمن قيمةً ما في المجموعة الممثَّلة. وكما في الشجرة ذات الارتفاع الثابت، تُمثِّل $cluster[i]$ القيمَ من $i\sqrt{u}$ إلى $(i+1)\sqrt{u}-1$، التي تكوِّن العنقودَ ذا الترتيب i.

في المستوى الأساسي، تُخزَّن عناصر المجموعات الديناميكية الحالية في بعض بنى $proto\text{-}vEB(2)$، وتُخزِّنُ بقيةُ بنى $proto\text{-}vEB(2)$ بتات الملخص. يَظهر في الشكل – تحت كلٍّ من البنى الأساسية التي لا تكوِّن ملخصًا – البتات التي تخزنها. فمثلاً، تخزِّن بنية $proto\text{-}vEB(2)$ التي عنوانها "العنصران 6 و 7" البتَ 6 في $A[0]$ (0، لأن العنصر 6 ليس من المجموعة)، وتخزِّن البت 7 في $A[1]$ (1، لأن العنصر 7 موجودٌ في المجموعة).

وكما في العناقيد، فإن كلَّ ملخص ليس سوى مجموعة ديناميكية حجم عالَمها $\sqrt{u}$، ولذلك نحن نمثِّل كلَّ ملخص على أنه بنية $proto\text{-}vEB(\sqrt{u})$. أما بتات الملخص الأربعة في البنية $proto\text{-}vEB(16)$ الأساسية، فموجودة في أقصى يسار بنية $proto\text{-}vEB(4)$ ، وتَظهر في النهاية في بنيتَي $proto\text{-}vEB(2)$. فمثلاً، في بنية $proto\text{-}vEB(2)$ التي عنوانها "العنقودان 2، 3" لدينا $A[0]=0$، وهذا يدل على أن العنقود 2 من البنية $proto\text{-}vEB(16)$ (الذي يتضمن العناصر 8, 9, 10, 11) كلُّه أصفار، ولدينا $A[1]=1$، وهذا يدل على أن العنقود 3 (الذي يتضمن العناصر 12, 13, 14, 15) فيه 1 واحد على الأقل. وتشير كل بنية $proto\text{-}vEB(4)$ إلى خلاصتها، والتي هي نفسها مخزنة على أنها بنية $proto\text{-}vEB(2)$. انظر، على سبيل المثال، إلى بنية $proto\text{-}vEB(2)$ الموجودة مباشرة إلى يسار تلك المسماة "العنصران 0، 1". لما كانت $A[0]$ هي 0، فهذا يعني أن بنية "العنصران 0، 1" كلها أصفار، ولما كانت $A[1]$ هي 1، فنحن نعلم أن بنية "العنصران 2، 3" تتضمن 1 واحد على الأقل.



الشكل 4.20 بنيةُ $proto\text{-}vEB(16)$ تمثلُ المجموعة $\{2,3,4,5,7,14,15\}$. تشيرُ البنيةُ إلى أربع بنى $proto\text{-}vEB(4)$ في العنقود $cluster[0..3]$، وإلى بنية ملخص، التي هي أيضًا $proto\text{-}vEB(4)$. كل بنيةٍ $proto\text{-}vEB(4)$ تشير إلى بنيتَيْ $proto\text{-}vEB(2)$ في عنقود $cluster[0..1]$، وإلى ملخص $proto\text{-}vEB(2)$. تتضمن كل بنية $proto\text{-}vEB(2)$ صفيفة $A[0..1]$ فقط مؤلفة من بتَّيْن. تخزِّن بنى $proto\text{-}vEB(2)$ الموضوعة فوق "العنصرين i,j" البتَّيْن i و j من المجموعة الديناميكية الحالية، وتخزِّن بنى $proto\text{-}vEB(2)$ الموضوعة فوق "العنقودين i، j" بتَّي الملخص للعنقودين i و j في البنية ذات المستوى الأعلى $proto\text{-}vEB(16)$. تشير الظلال الغامقة – زيادةً في الإيضاح – إلى المستوى الأعلى من بنية proto-vEB التي تخزِّن معلومات الملخص لبنية أبيها؛ وفيما عدا ذلك، تكون بنية proto-vEB هذه مطابقة لأية بنية proto-vEB أخرى لها حجم العالَم ذاته.

2.2.20 العمليات على بنية proto van Emde Boas

نصف الآن كيفية إنجار العمليات على بنية proto-vEB. نفحص أولاً عمليات الاستعلام – MEMBER و MINIMUM و SUCCESSOR – التي لا تغيِّر بنيةَ proto-vEB. ثم نناقش بعدها INSERT و DELETE.

وسنترك MAXIMUM و PREDECESSOR – اللتين تناظران MINIMUM و SUCCESSOR على الترتيب – إلى التمرين 20.2-1.

تأخذ كلٌّ من العمليات: MEMBER و SUCCESSOR و PREDECESSOR و INSERT و DELETE موسطًا x، إضافة إلى بنية proto-vEB V، حيث تفترض كلٌّ من هذه العمليات أن $0 \le x < V.u$.

كيف نحدِّد وجود قيمة في المجموعة

لإنجاز MEMBER(x) نحتاج إلى العثور على البت الموافق لـ x ضمن بنية $proto\text{-}vEB(2)$ المناسبة. ويمكننا إجراء ذلك بزمن $O(\lg\lg u)$، وذلك بالمرور على بنى $summary$ جميعها. يأخذ الإجراء التالي بنيةَ $proto\text{-}v$ V وقيمةً ما x، ويعيد بتًّا يدل على وجود x في المجموعة الديناميكية التي تمثلها V.

```
PROTO-vEB-MEMBER(V, x)
1  if V.u == 2
2      return V.A[x]
3  else return PROTO-vEB-MEMBER(V.cluster[high(x)], low(x))
```

يعمل الإجراء PROTO-vEB-MEMBER على النحو الآتي: يختبر السطرُ 1 الحالة الأساسية، حيث V هي بنية $proto\text{-}vEB(2)$. ويعالج السطرُ 2 الحالة الأساسية، وذلك بإعادة البت المناسب من الصفيفة A. يتعامل السطر 3 مع الحالة العودية، "نزولاً" باتجاه أصغر بنية proto-vEB مناسبة. تبين القيمة $\text{high}(x)$ أيَّة بنية $proto\text{-}vEB(\sqrt{u})$ نزور، وتحدِّد $\text{low}(x)$ أيَّ عنصرٍ ضمن بنية $proto\text{-}vEB(\sqrt{u})$ نستعلم.

لننظر ماذا يحدث عندما نستدعي PROTO-vEB-MEMBER$(V, 6)$ على بنية $proto\text{-}vEB(16)$ في الشكل 20.4. لما كانت $\text{high}(6) = 1$ عندما تكون $u = 16$، فإننا نقوم بإجراء العودية ضمن بنية $proto\text{-}vEB(4)$ في أعلى اليمين، ونسأل عن العنصر $\text{low}(6) = 2$ في تلك البنية. في هذا الاستدعاء العودي يكون $u = 4$، ولذلك نقوم بإجراء العودية مرة أخرى. ولما كانت $u = 4$، فلدينا $\text{high}(2) = 1$، و $\text{low}(2) = 0$، ولذلك نسأل عن العنصر 0 من بنية $proto\text{-}vEB(2)$ في أعلى اليمين. يتبيَّن أن هذا الطلب العودي هو حالة أساسية، ولذلك يعيد $A[0] = 0$ صعودًا عبر سلسلة الاستدعاءات العودية. وهكذا، نستنتج أن PROTO-vEB-MEMBER$(V, 6)$ يعيد 0، وهذا يعني أن 6 ليس من المجموعة.

وبغية تحديد زمن تنفيذ PROTO-vEB-MEMBER، نرمز بـ $T(u)$ إلى زمن تنفيذه على بنية $proto\text{-}vEB(u)$. إنَّ كلَّ استدعاءٍ عودي يستغرق زمنًا ثابتًا، لا يتضمن الزمن الذي تتطلبه الاستدعاءات العودية التي يقوم بها. عندما يقوم PROTO-vEB-MEMBER باستدعاء عودي، فإن الاستدعاء يكون على بنية $proto\text{-}vEB(\sqrt{u})$. وهكذا يمكننا توصيف زمن التنفيذ بالمعادلة التكرارية $T(u) = T(\sqrt{u}) + O(1)$، التي شاهدناها آنفًا في المعادلة التكرارية (20.2). وحلُّ هذه المعادلة هو $T(u) = O(\lg\lg u)$، وبذلك

نستنتج أن PROTO-vEB-MEMBER يُنفذ بزمنٍ $O(\lg\lg u)$.

كيف نَجِدُ العنصرَ الأصغري

نبحث الآن في كيفية إنجاز عملية MINIMUM. يعيد الإجراءُ PROTO-vEB-MINIMUM(V) أصغرَ عنصرٍ في بنية proto-vEB V، أو يعيد NIL إذا كان V يمثِّلُ مجموعةً خالية.

```
PROTO-vEB-MINIMUM(V)
 1  if V.u == 2
 2      if V.A[0] == 1
 3          return 0
 4      elseif V.A[1] == 1
 5          return 1
 6      else return NIL
 7  else min-cluster = PROTO-vEB-MINIMUM(V.summary)
 8      if min-cluster == NIL
 9          return NIL
10      else offset = PROTO-vEB-MINIMUM(V.cluster[min-cluster])
11          return index(min-cluster, offset)
```

يعمل هذا الإجراء كما يلي. يختبر السطر 1 الحالة الأساسية، والتي تعالجها الأسطر 2-6 بقوة ضاربة brute-force. تعالج الأسطر 7-11 الحالة العودية. أولاً، يَجِدُ السطرُ 7 رقمَ أول عنقود يتضمن عنصرًا من المجموعة. يقوم بذلك باستدعاء PROTO-vEB-MINIMUM عوديًّا على $V.summary$، وهو بنية $proto\text{-}vEB(\sqrt{u})$. يُسنِدُ السطرُ 7 رقمَ العنقود هذا إلى المتحول *min-cluster*. فإذا كانت المجموعة خالية، يعيدُ الاستدعاء العودي القيمةَ NIL، ويعيد السطرُ 9 القيمة NIL. وإلا، يكون أصغر عنصر في المجموعة موجودًا في مكان ما في العنقود رقم *min-cluster*. يَجِدُ الاستدعاءُ العودي في السطر 10 الانزياحَ offset ضمن عنقود أصغر عنصر في هذا العنقود. وفي النهاية، يبني السطرُ 11 قيمةَ أصغر عنصر انطلاقًا من رقم العنقود والانزياح، ويعيدُ هذه القيمة.

ومع أن الاستعلام ضمن معلومة الملخص يسمحُ بالعثور على العنقود الذي يتضمن أصغر عنصر بسرعة، إلا أن هذا الإجراء لا يُنفَّذ بزمن $O(\lg\lg u)$ في أسوأ الحالات، لأنه يقوم باستدعاءين عوديين على بنى $proto\text{-}vEB(\sqrt{u})$. نرمز إلى زمن تنفيذ PROTO-vEB-MINIMUM في أسوأ الحالات، على بنية $proto\text{-}vEB(u)$، بالرمز $T(u)$، فتكون لدينا المعادلة التكرارية:

$$T(u) = 2T(\sqrt{u}) + O(1) \ . \qquad (3.20)$$

نستخدم، مرة أخرى، تغيير المتحولات لحل هذه المعادلة، حيث نجعل $m = \lg u$، وهذا يعطي

$$T(2^m) = 2T(2^{m/2}) + O(1) \ .$$

وبتسمية $S(m) = T(2^m)$، نحصل على

$$S(m) = 2S(m/2) + O(1) ,$$

التي حلها، وفق الحالة 1 من الطريقة العامة، هو $S(m) = \Theta(m)$. وبإعادة تغيير $S(m)$ إلى $T(u)$، يكون لدينا $T(u) = T(2^m) = S(m) = \Theta(m) = \Theta(\lg u)$. وبذلك، نرى أنه بسبب الاستدعاء العودي الثاني ينفَّذ PROTO-vEB-MINIMUM بزمن $\Theta(\lg u)$، وليس بالزمن المطلوب $O(\lg \lg u)$.

كيف نَجِدُ العنصرَ التالي

إن عملية SUCCESSOR هي أشدُّ سوءًا مما سبقها. في أسوأ الحالات، يقوم الإجراء باستدعاءين عوديين، إضافة إلى استدعاء PROTO-vEB-MINIMUM. ويعيد الإجراء PROTO-vEB-MINIMUM(V, x) أصغر العناصر في بنية proto-vEB V الذي هو أكبر من x، أو يعيدُ NIL إذا لم يوجد عنصر في V أكبرَ من x. لا يتطلب هذا الإجراء أن يكون x عنصرًا member في المجموعة، ولكنه يفترض أن يكون $0 \le x < V.u$.

```
PROTO-vEB-SUCCESSOR(V, x)
 1  if V.u == 2
 2      if x == 0 and V.A[1] == 1
 3          return 1
 4      else return NIL
 5  else offset = PROTO-vEB-SUCCESSOR(V.cluster[high(x)], low(x))
 6      if offset ≠ NIL
 7          return index(high(x), offset)
 8      else  succ-cluster = PROTO-vEB-SUCCESSOR(V.summary, high(x))
 9          if  succ-cluster == NIL
10              return NIL
11      else offset = PROTO-vEB-MINIMUM (V.cluster[succ-cluster])
12          return index(succ-cluster, offset)
```

يعمل الإجراء PROTO-vEB-SUCCESSOR على النحو الآتي: يختبر السطر 1، كالعادة، الحالةَ الأساسية، التي تعالجها الأسطر 2-4 بالقوة الضاربة: الطريقة الوحيدة التي يمكن أن يكون فيها للعنصر x عنصرٌ تالٍ ضمن بنية *proto-vEB*(2) هي عندما تكون $x = 0$ وتكون قيمة $A[1]$ هي 1. تعالجُ الأسطر 5-12 الحالةَ العودية. يبحث السطر 5 عن عنصرٍ تالٍ لـ x ضمن عنقود x، مسندًا النتيجة إلى $offset$. يُحدِّد السطرُ 6 وجودَ عنصرٍ تالٍ لـ x ضمن عنقوده؛ فإذا كان له عنصرٌ تالٍ، يحسب السطر 7 قيمةَ هذا العنصر ويعيدها. وإلا، علينا أن نبحث في عناقيد أخرى. يسندُ السطرُ 8 رقمَ العنقودِ التالي غير الخالي إلى *succ-cluster*، مستخدمًا معلومات الملخص لإيجاده. يختبرُ السطرُ 9 مطابقةَ قيمةِ *succ-cluster* لـ NIL، ويعيد السطرُ 10 القيمةَ NIL إذا كانت العناقيد التالية جميعها خالية. إذا لم تكن قيمةُ

هي NIL، يُسنِدُ السطرُ 11 إلى $offset$ أولَ عنصرٍ ضمن هذا العنقود، ويَحسبُ السطرُ 12 أصغرَ عنصرٍ في هذا العنقود ويعيده. $succ\text{-}cluster$

في أسوأ الحالات، يستدعي PROTO-vEB-SUCCESSOR نفسَه عوديًّا مرتين على بنى $proto\text{-}vEB(\sqrt{u})$، ويستدعي PROTO-vEB-MINIMUM مرةً واحدةً على بنية $proto\text{-}vEB(\sqrt{u})$. وبذلك، تكون المعادلة التكرارية الخاصة بزمن تنفيذ PROTO-vEB-SUCCESSOR بأسوأ الحالات هي $T(u)$:

$$\begin{aligned} T(u) &= 2T(\sqrt{u}) + \Theta(\lg \sqrt{u}) \\ &= 2T(\sqrt{u}) + \Theta(\lg u) \ . \end{aligned}$$

يمكننا استخدام التقنية نفسها التي استخدمناها في المعادلة التكرارية (1.20) لنبين أن حل هذه المعادلة التكرارية هو $T(u) = \Theta(\lg u \lg\lg u)$. وهكذا، فإن PROTO-vEB-SUCCESSOR أبطأ تقاربيًّا من PROTO-vEB-MINIMUM.

إدراج عنصر

لإدراج عنصر، نحن بحاجة إلى أن يكون إدراجه في العنقود الملائم وإلى وضع القيمة 1 في بت الملخص لهذا العنقود. يُدرِج الإجراءُ PROTO-vEB-INSERT(V, x) القيمةَ x في بنية proto-vEB V.

```
PROTO-vEB-INSERT(V, x)
1  if V.u == 2
2      V.A[x] = 1
3  else PROTO-vEB-INSERT(V.cluster[high(x)], low(x))
4      PROTO-vEB-INSERT(V.summary, high(x))
```

في الحالة الأساسية، يضعُ السطرُ 2 القيمةَ 1 في البت الملائم في الصفيفة A. في الحالة العودية، يُدْرِجُ الاستدعاءُ العودي، في السطر 3، x في العنقود الملائم، ويضعُ السطرُ 4 القيمةَ 1 في بت الملخص لهذا العنقود.

ولما كان الإجراء PROTO-vEB-INSERT يقوم باستدعاءين عوديين في أسوأ الحالات، فإن المعادلة التكرارية (3.20) تَصِفُ زمنَ تنفيذ هذا الإجراء. لذلك، يُنفَّذ PROTO-vEB-INSERT بزمن $\Theta(\lg u)$.

حذف عنصر

إن عملية DELETE أعقد من الإدراج. لأنه إذا كان بإمكاننا دومًا وضع القيمة 1 في بت الملخص عند الإدراج، فإننا لا نستطيع دومًا إعادة وضع القيمة 0 في بت الملخص نفسه عند الحذف. ونحن بحاجةٍ إلى تحديد: هل تساوي قيمةُ أحد البتات في العنقود الملائم القيمةَ 1؟ حسب تعريفنا لبنى proto-vEB، علينا أن نفحص جميع البتات التي عددها $\sqrt{u}$ ضمن عنقود لنحدِّد: هل يساوي أحدُها القيمةَ 1؟ ثمة حلٌّ بديل، وهو أن نضيف واصفة n إلى بنية proto-vEB، وتَعُدُّ عدد العناصر في البنية. سنترك تنجيز

PROTO-vEB-DELETE إلى التمرينين 2.20-2 و 3-2.20.

من الواضح أن علينا تعديل بنية proto-vEB لتخفيض كل عملية بحيث يكون فيها استدعاء عودي واحد على الأكثر. سنرى في المقطع التالي كيف يجري ذلك.

تمارين

1-2.20

اكتب شبه رماز للإجراءين PROTO-vEB-MAXIMUM و PROTO-vEB-PREDECESSOR.

2-2.20

اكتب شبه رماز للإجراء PROTO-vEB-DELETE. يجب أن يحدِّثَ بِتّ الملخص الملائم بمسح البتات المرتبطة ضمن العنقود. ما هو زمن التنفيذ في أسوأ الحالات لإجرائك؟

3-2.20

أضف الواصفة n إلى كل بنية proto-vEB، التي تعطي عدد العناصر الموجودة حاليًا في المجموعة التي تمثّلها، واكتب شبه رماز للإجراء PROTO-vEB-DELETE الذي يستخدم الواصفة n ليحدد متى يضع القيمة 0 في بتات الملخص. ما هو زمن تنفيذ الحالة الأسوأ لإجرائك؟ ما هي الإجراءات الأخرى التي تحتاج إلى تغيير بسبب هذه الواصفة الجديدة؟ هل تؤثر هذه التغييرات على أزمان تنفيذها؟

4-2.20

عدِّل بنية proto-vEB لتدعم المفاتيح المكررة.

5-2.20

عدِّل بنية proto-vEB لتدعم المفاتيح التي لها معطيات تابعة مرفقة.

6-2.20

اكتب شبه رماز لإجراءٍ ينشئ بنية $proto\text{-}vEB(u)$.

7-2.20

أثبت أنه إذا نفِّذ السطر 9 من PROTO-vEB-MINIMUM، فإن بنية proto-vEB تكون خالية.

8-2.20

افترض أننا صممنا بنية proto-vEB بحيث أن كل صفيفة $cluster$ فيها تتضمن $u^{1/4}$ عنصرًا فقط. ماذا ستكون أزمان تنفيذ كلٍّ من هذه العمليات؟

3.20 شجرة van Emde Boas

إن بنية proto-vEB المعروضة في المقطع السابق قريبة لما نرغبُ بتحقيقه من حيث أزمان تنفيذ $O(\lg\lg u)$. لكنها لا تفي بالغرض، لأننا يجب أن نقوم بالكثير من الاستدعاءات العودية في معظم العمليات. سنصمِّم في هذا المقطع بنيةَ معطياتٍ شبيهةً ببنية proto-vEB لكنها تخزِّن معلوماتٍ أقلَّ بقليل، وبذلك تنتفي الحاجة إلى بعض الاستدعاءات العودية.

لاحظنا في المقطع 2.20، أن الفرضية التي وضعناها بخصوص حجم العالَم – أي $u = 2^{2^k}$، حيث k عدد صحيح – مقيَّدة جدًّا، وهذا يجعل القيمَ الممكنة لـ u مجموعة محدودة جدًّا. لذلك سنسمح، بدءًا من الآن، بأن يكون حجم العالَم u أية قوة صحيحة من 2، وعندما لا تكون $\sqrt{u}$ عددًا صحيحًا – أي، إذا كانت u قوة فردية لـ 2 ($u = 2^{2k+1}$، حيث $k \geq 0$ عدد صحيح) – فإننا سنقسم البتات التي عددها $\lg u$ لعدد ما إلى البتات $\lceil(\lg u)/2\rceil$ الأكثر أهمية، والبتات $\lfloor(\lg u)/2\rfloor$ الأقل أهمية. وللتبسيط، نرمز لـ $2^{\lceil(\lg u)/2\rceil}$ (أي "الجذر التربيعي الأعلى" لـ u) بالرمز $\sqrt[\uparrow]{u}$ ولـ $2^{\lfloor(\lg u)/2\rfloor}$ (أي "الجذر التربيعي الأدنى" لـ u) بالرمز $\sqrt[\downarrow]{u}$، فيكون لدينا $u = \sqrt[\uparrow]{u} \cdot \sqrt[\downarrow]{u}$، وعندما يكون u قوةً زوجيةً لـ 2 ($u = 2^{2k}$، حيث k عدد صحيح)، يكون لدينا $\sqrt[\uparrow]{u} = \sqrt[\downarrow]{u} = \sqrt{u}$. وحيث إننا سمحنا بأن تكون u قوةً فرديةً لـ 2، فيجب إعادة تعريف دوالنا المساعدة المذكورة في المقطع 2.20:

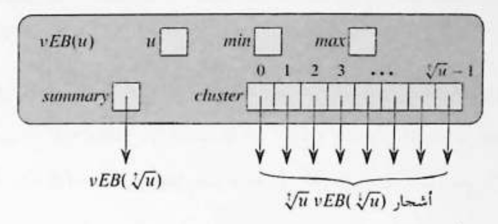
$$
\begin{aligned}
\text{high}(x) &= \lfloor x/\sqrt[\downarrow]{u} \rfloor \,, \\
\text{low}(x) &= x \bmod \sqrt[\downarrow]{u} \,, \\
\text{index}(x, y) &= x\sqrt[\downarrow]{u} + y \,.
\end{aligned}
$$

1.3.20 أشجار van Emde Boas

تعدِّل **شجرة *van Emde Boas***، أو **شجرة *vEB***، بنيةَ proto-vEB. نرمز لشجرة vEB حجم عالَمها u بالرمز $vEB(u)$، وما لَم تكن u مساويةً الحجم الأساسي 2، فإن الواصفةَ *summary* تشير إلى شجرة $vEB(\sqrt[\uparrow]{u})$، وتشير الصفيفةُ $cluster[0\,..\sqrt[\uparrow]{u}-1]$ إلى $\sqrt[\uparrow]{u}$ $vEB(\sqrt[\downarrow]{u})$ شجرة. وكما هو موضَّح في الشكل 5.20، تتضمنُ شجرةُ vEB واصفتَيْن غير موجودتين في بنية proto-vEB:

- تخزِّن min أصغرَ عناصر شجرة vEB، و
- تخزِّن max أكبرَ عناصر شجرة vEB.

إضافة إلى ذلك، لا يَظهر العنصرُ المخزَّن في min في أيٍّ من الأشجار العودية التي عددها $vEB(\sqrt[\downarrow]{u})$ التي تشير إليها الصفيفة *cluster*. لذلك، فإن عدد العناصر المخزَّنة في شجرة $vEB(u)$ V هو $V.min$ إضافة إلى جميع العناصر المخزنة عوديًّا في الأشجار التي عددها $vEB(\sqrt[\downarrow]{u})$ والتي يشير إليها



الشكل 5.20 المعلومات في شجرة $vEB(u)$ عندما تكون $u > 2$. تتضمن البنيةُ حجمَ العالَم u، والعنصرين min و max، ومؤشرًا $summary$ على شجرة $vEB(\sqrt[\uparrow]{u})$، وصفيفة $cluster[0..\sqrt[\uparrow]{u}-1]$ من $\sqrt[\uparrow]{u}$ مؤشرًا على $vEB(\sqrt[\downarrow]{u})$ شجرة.

$V.cluster[0..\sqrt[\uparrow]{u}-1]$. لاحظ أننا نعامل min و max معاملةً مختلفة عندما تتضمن شجرة vEB عنصرين أو أكثر: العنصر المخزَّن في min لا يَظهر في أيٍّ من العناقيد، في حين يَظهر العنصرُ المخزَّن في max.

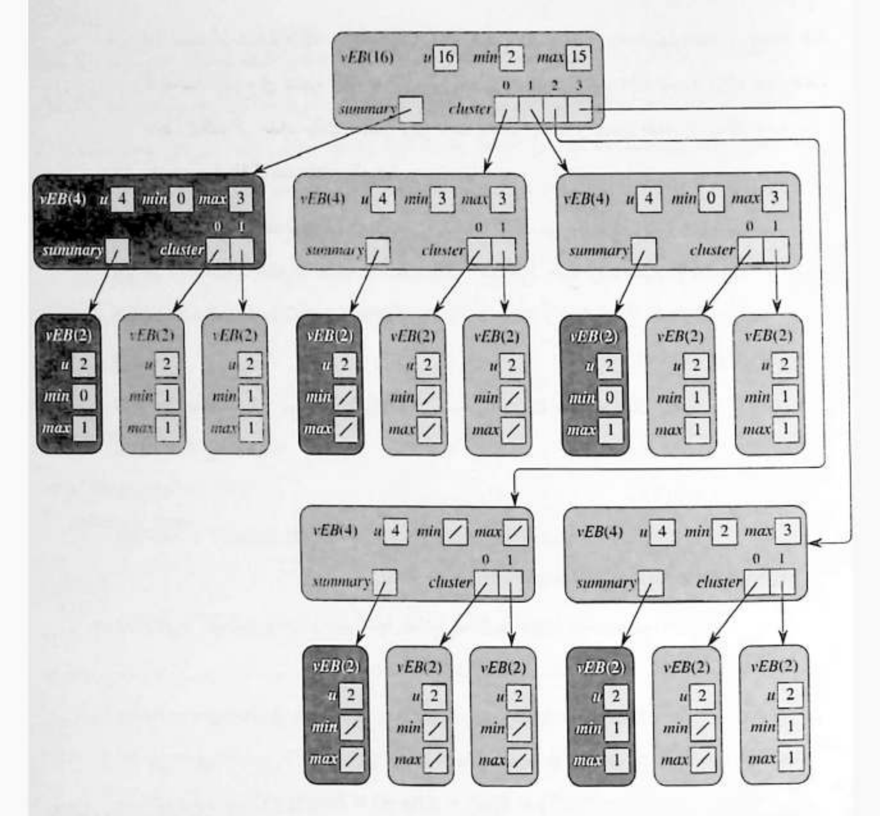
ولما كان الحجم الأساسي هو 2، فإن شجرة $vEB(2)$ لا تحتاج إلى الصفيفة A التي كانت لدى بنية $proto\text{-}vEB(2)$ الموافقة. وبدلاً من ذلك، يمكننا تحديد عناصرها من واصفتَيْها min و max. فإذا كانت شجرة vEB خاليةً من العناصر – بصرف النظر عن حجم عالَمها u – فإن قيمة كلٍّ من min و max هي NIL.

يبين الشكل 6.20 شجرة $vEB(16)$ V تتضمن المجموعة $\{2,3,4,5,7,14,15\}$. ولما كان أصغر العناصر هو 2، فإن $V.min$ تساوي 2، وحتى لو كان $\text{high}(2)=0$، فإن العنصر 2 لا يَظهر في الشجرة $vEB(4)$ التي يشير إليها $V.cluster[0]$: لاحظ أن $V.cluster[0].min$ يساوي 3، وبهذا لا يكون العنصر 2 في شجرة vEB. وبالمثل، لما كانت قيمة $V.cluster[0].min$ تساوي 3، وكان 2 و 3 هما العنصران الوحيدان في $V.cluster[0]$، فإن العنقودين $vEB(2)$ ضمن $V.cluster[0]$ خاليان.

سيتبيَّن لاحقًا أن الواصفتَيْن min و max لهما دورٌ أساسيٌّ في تخفيض عدد الاستدعاءات العودية ضمن العمليات على أشجار vEB. ستساعدنا هاتان الواصفتان على أربعة صعد:

1. لم تعد العمليتان MINIMUM و MAXIMUM بحاجة إلى عودية، لأن بإمكانهما إعادة قيم min و max فقط.

2. يمكن أن تتجنب العمليةُ SUCCESSOR إجراءَ استدعاءٍ عوديٍّ لتحديد: هل العنصر التالي لقيمة ما x موجودٌ ضمن $\text{high}(x)$؟ وذلك لأن العنصر التالي لـ x يقع ضمن عنقودها إذا وفقط إذا كانت x أصغر تمامًا من الواصفة max لعنقودها. يمكن تطبيق برهان مشابه على PREDECESSOR و min.



الشكل 6.20 شجرة $vEB(16)$ موافقة لشجرة proto-vEB الموجودة في الشكل 4.20. تخزِّن هذه الشجرة المجموعة $\{2, 3, 4, 5, 7, 14, 15\}$. تدل '/' على قيم NIL. لا تَظهر القيمة المخزنة في الواصفة min لشجرة vEB في أيٍّ من عناقيدها. استُعملت الظلالُ الغامقة هنا لنفس الغرض المذكور في الشكل 4.20.

3. يمكننا بزمن ثابت معرفة: هل الشجرة vEB خالية، أم أنها تحتوي على عنصر وحيد، أم أنها تحتوي على عنصرين على الأقل؟ وذلك بالاعتماد على قيمتَي min و max. وسنستفيد من هذه الإمكانية في عمليتَي INSERT و DELETE. فإذا لم تكن min و max تساويان NIL، فإن شجرة vEB تكون خاليةً من العناصر. وإذا كانت min و max لا تساويان NIL، ولكنهما متساويتان، فإن شجرة vEB تتضمن عنصرًا واحدًا تمامًا. وإذا كانت min و max لا تساويان NIL، ولكنهما غير متساويتين، فإن شجرة vEB تتضمن عنصرين أو أكثر.

4. إذا علمنا أن شجرة vEB خالية، يمكننا إدراج عنصر فيها بتعديل واصفتَيْها min و max فقط. ومن ثَم يمكننا الإدراج في شجرة vEB خالية بزمن ثابت. وبالمثل، إذا علمنا أن شجرة vEB فيها عنصر واحد فقط، يمكننا أن نحذف ذلك العنصر بزمن ثابت بتعديل min و max فقط. ستمكننا هذه الخواص من تخفيض سلسلة الاستدعاءات العودية.

فإذا كان حجم العالَم u قوةً فرديةً للعدد 2، فإن الفرق بين حجمَيْ شجرة vEB الملخص والعناقيد لن يؤثر في أزمنة التنفيذ التقاربية لعمليات شجرة vEB. وستكون جميع أزمنة تنفيذ الإجراءات العودية التي تنجز عمليات شجرة vEB موصَّفة بالمعادلة التكرارية.

$$T(u) \leq T(\sqrt[\uparrow]{u}) + O(1) . \qquad (4.20)$$

تشبه هذه المعادلة التكرارية المعادلة (2.20)، وسنحلها بطريقة مشابهة. ليكن لدينا $m = \lg u$، نعيد كتابة المعادلة التكرارية بالصيغة:

$$T(2^m) \leq T(2^{\lceil m/2 \rceil}) + O(1) .$$

وبملاحظة أن $\lceil m/2 \rceil \leq 2m/3$ لجميع قيم $m \geq 2$، يكون لدينا:

$$T(2^m) \leq T(2^{2m/3}) + O(1) .$$

وبافتراض $S(m) = T(2^m)$، نعيد كتابة المعادلة التكرارية الأخيرة هذه بالصيغة:

$$S(m) \leq S(2m/3) + O(1) ,$$

وحلُّها – وفق الحالة 2 من الطريقة الأساسية – هو $S(m) = O(\lg m)$. (ليس هناك فارق – في الحل التقاربي – بين الكسرين 2/3 و 1/2، لأننا عندما نطبق الطريقة الأساسية، نجد أن $\log_{3/2} 1 = \log_2 1 = 0$.) وبذلك، يكون لدينا $T(u) = T(2^m) = S(m) = O(\lg m) = O(\lg \lg u)$.

قبل استخدام شجرة van Emde Boas، علينا معرفة حجم العالَم u، بحيث نستطيع إنشاء شجرة van Emde Boas بالحجم الملائم الذي يمثِّل في البداية مجموعةً خالية. يُطلب في المسألة 20-1 برهان أن الحجم الكلي المطلوب لشجرة van Emde Boas هو $O(u)$، وأنه يمكن إنشاء شجرة خالية مباشرة بزمن $\Theta(u)$. وبالمقابل، يمكننا إنشاء شجرة حمراء-سوداء خالية بزمن ثابت. لذلك، ربما لا نرغب في استخدام شجرة van Emde Boas عندما نجري عددًا صغيرًا فقط من العمليات، لأن زمن إنشاء بنية المعطيات قد يتجاوز الزمن الذي نكسبه في العمليات المنفصلة. هذه السيئة ليست هامة، لأننا نستخدم عادة بنيةَ معطياتٍ بسيطةً، مثل صفيفة أو قائمة مترابطة، لتمثيل مجموعةٍ عددُ عناصرها قليل.

2.3.20 العمليات على شجرة van Emde Boas

نحن الآن جاهزون لمعرفة كيفية إنجاز العمليات على شجرة van Emde Boas. سندرس، كما فعلنا في بنية proto van Emde Boas، عمليات الاستعلامات أولاً، ثم INSERT و DELETE. ونظرًا لعدم التناظر الطفيف

بين العنصرين الأصغر والأعظم في شجرة vEB – عندما تتضمن شجرة vEB عنصرين على الأقل، لا يَظهر العنصر الأصغر ضمن عنقود، في حين يَظهر العنصر الأعظم – سنورد شبه رماز لجميع عمليات الاستعلام الخمس. وكما في العمليات على بنى proto van Emde Boas، *تفترض* العمليات التي تأخذ موسطين V و x، أن $0 \leq x < V.u$، حيث V هي شجرة van Emde Boas و x هو عنصر.

كيف نَجِدُ العنصرين الأصغري والأعظمي

لما كنا نخزِّن العنصرين الأصغري والأعظمي في الواصفتين min و max، فهنالك عمليتان مؤلَّفتان من سطر واحد، تأخذان زمنًا ثابتًا:

```
vEB-TREE-MINIMUM(V)
1  return V.min
```

```
vEB-TREE-MAXIMUN(V)
1  return V.max
```

تحديدُ وجودِ قيمةٍ ما u في المجموعة

للإجراء vEB-TREE-MEMBER(V, x) حالةٌ عودية كتلك الموجودة في PROTO-vEB-MEMBER، لكن الحالةَ الأساسيةَ مختلفةٌ قليلاً. لذا فإننا نفحص مباشرة المساواة بين x والعنصر الأصغري أو الأعظمي. وحيث إن شجرة vEB لا تخزِّن البتات مثلما تفعل بنية proto-vEB، فإننا نصمِّم vEB-TREE-MEMBER للحصول على TRUE أو FALSE بدلاً من 1 أو 0.

```
vEB-TREE-MEMBER(V, x)
1  if x == V.min or x == V.max
2       return TRUE
3  elseif V.u == 2
4       return FALSE
5  else return vEB-TREE-MEMBER(V.cluster[high(x)], low(x))
```

يفحص السطر 1 المساواة بين x والعنصر الأصغري أو الأعظمي. فإذا تحققت، فإن السطر 2 يعيد القيمةَ TRUE. وإلا، يختبر السطرُ 3 الحالة الأساسية. ولما كانت شجرة $vEB(2)$ لا تتضمن سوى العناصر الموجودة في min و max، إذا كانت هذه هي الحالةَ الأساسية، فإن السطر 4 يُعيد القيمة FALSE. تجري معالجة الاحتمال الآخر – إذا لم تكن هذه هي الحالة الأساسية، وكانت x لا تساوي min ولا max – بالاستدعاء العودي في السطر 5.

توصِّف المعادلة التكرارية (4.20) زمنَ تنفيذ إجراء vEB-TREE-MEMBER، لذا فإنه يستغرق زمنًا $O(\lg \lg u)$.

كيف نَجِدُ اللاحق والسابق

نوضح فيما يلي كيف ننجز عملية SUCCESSOR. تذكَّر أن إجراء PROTO-vEB-SUCCESSOR(V, x) قد يُجري استدعاءَيْن عوديين: أحدهما لتحديد: هل لاحق x موجود في عنقود x نفسه؟ وإذا لم يكن كذلك، فآخر للعثور على العنقود الذي يتضمن لاحق x. ولما كان باستطاعتنا الوصول سريعًا إلى القيمة العظمى في شجرة vEB، فيمكننا تجنب إجراء استدعاءين عوديين، والقيام بدلاً من ذلك باستدعاءٍ عودي واحد على عنقود أو على الملخص، ولكن ليس عليهما معًا.

```
vEB-TREE-SUCCESSOR(V, x)
 1  if V.u == 2
 2      if x == 0 and V.max == 1
 3          return 1
 4      else return NIL
 5  elseif V.min ≠ NIL and x < V.min
 6      return V.min
 7  else max-low = vEB-TREE-MAXIMUM(V.cluster[high(x)])
 8      if max-low ≠ NIL and low(x) < max-low
 9          offset = vEB-TREE-SUCCESSOR(V.cluster[high(x)], low(x))
10          return index(high(x), offset)
11      else succ-cluster = vEB-TREE-SUCCESSOR(V.summary, high(x))
12          if succ-cluster == NIL
13              return NIL
14          else offset = vEB-TREE-MINIMUM(V.cluster[succ-cluster])
15              return index(succ-cluster, offset)
```

لهذا الإجراء ستة تعليمات **return**، وحالاتٌ متعددة. نبدأ بالحالة الأساسية في الأسطر 2–4، التي تعيد القيمة 1 في السطر 3 إذا كنا نحاول إيجاد لاحق 0 وكان 1 موجودًا في المجموعة المؤلفة من عنصرين؛ وإلا، تعيد الحالة الأساسية القيمة NIL في السطر 4.

إذا لَم نكن في الحالة الأساسية، فإننا نتفقَّد بعد ذلك في السطر 5: هل x أصغر تمامًا من أصغر عنصر؟ فإذا كان كذلك، نعيد ببساطة أصغر عنصر في السطر 6.

إذا وصلنا إلى السطر 7، نعرف عندها أننا لسنا في الحالة الأساسية، وأن x أكبر أو يساوي أصغر قيمة في شجرة vEB V. يسندُ السطرُ 7 قيمةَ أعظم عنصر في عنقود x إلى *max-low*. إذا كان عنقود x يتضمن عنصرًا أكبر من x، نعلم عندها أن لاحقَ x موجودٌ في مكانٍ ما ضمن عنقود x. يَختبر السطرُ 8 هذا الشرط. إذا كان لاحق x موجودًا في عنقود x، يحدِّدُ السطرُ 9 مكانه في العنقود، ويعيد السطرُ 10 العنصرَ اللاحق بنفس طريقة السطر 7 من PROTO-vEB-SUCCESSOR.

نصل إلى السطر11 إذا كان x أكبر أو يساوي أعظم عنصر في عنقوده. في هذه الحالة، تَجِدُ الأسطرُ 11-15 لاحقَ x بنفس الطريقة المعروضة في السطور 8-12 من PROTO-vEB-SUCCESSOR.

من السهل رؤية كيف توصِّف المعادلة التكرارية (4.20) زمن تنفيذ vEB-TREE-SUCCESSOR. واعتمادًا على نتيجة الاختبار في السطر 8، يستدعي الإجراءُ نفسَه عوديًّا إما في السطر 9 (على شجرة vEB وبحجم عالَم $\sqrt[\downarrow]{u}$) وإما في السطر 11 (على شجرة vEB بحجم عالَم $\sqrt[\uparrow]{u}$). في كلتا الحالتين، يكون الاستدعاء العودي الوحيد هو على شجرة vEB وبحجم عالَم $\sqrt[\uparrow]{u}$ على الأكثر. يستغرقُ باقي الإجراء زمنًا $O(1)$، ومن ضمنه الاستدعاءان vEB-TREE-MINIMUM و vEB-TREE-MAXIMUM. لذلك، يُنفَّذ vEB-TREE-SUCCESSOR بزمن $O(\lg \lg u)$ في أسوأ الحالات.

إن إجراء vEB-TREE-PREDECESSOR مناظرٌ لإجراءِ vEB-TREE-SUCCESSOR، ولكن مع حالة إضافية.

```
vEB-TREE-PREDECESSOR(V, x)
 1  if V.u == 2
 2      if x == 1 and V.min == 0
 3          return 0
 4      else return NIL
 5  elseif V.max ≠ NIL and x > V.max
 6      return V.max
 7  else min-low = vEB-TREE-MINIMUM(V.cluster[high(x)])
 8      if min-low ≠ NIL and low(x) > min-low
 9          offset = vEB-TREE-PREDECESSOR(V.cluster[high(x)], low(x))
10          return index(high(x), offset)
11      else pred-cluster = vEB-TREE-PREDECESSOR(V.summary, high(x))
12          if pred-cluster == NIL
13              if V.min ≠ NIL and x > V.min
14                  return V.min
15              else return NIL
16          else offset = vEB-TREE-MAXIMUM(V.cluster[pred-cluster])
17              return index(pred-cluster, offset)
```

يكوِّن السطران 13–14 الحالةَ الإضافية. تحدث هذه الحالة عندما لا يكون لاحقُ x – إن كان موجودًا – ضمن عنقود x. تأكَّد لدينا في إجراء vEB-TREE-SUCCESSOR أنه إذا كان لاحقُ x موجودًا خارج عنقود x، فلا بد من أن يوجد في عنقودٍ ذي رقم أعلى. لكنْ إذا كان سابقُ x هو القيمة الصغرى في شجرة V vEB، فإن السابق لا يوجَد في أيٍّ من العناقيد. يتفقد السطرُ 13 هذا الشرط، ويعيد السطرُ 14 القيمةَ الصغرى كما هو مطلوب.

لا تُؤثر هذه الحالة الإضافية على زمن التنفيذ المقارب لزمن إجراء vEB-TREE-PREDECESSOR، عند مقارنته بـ vEB-TREE-SUCCESSOR، وهكذا ينفّذ vEB-TREE-PREDECESSOR بزمنٍ $O(\lg \lg u)$ في أسوأ الحالات.

إدراج عنصر

نبحث الآن في كيفية إدراج عنصر ضمن شجرة vEB. نذكّر هنا بأن PROTO-vEB-INSERT أجرى استدعاءين عوديين: أحدهما لإدراج العنصر والآخر لإدراج رقم عنقود العنصر ضمن الملخص. سيقوم إجراء vEB-TREE-INSERT باستدعاء عودي واحد فقط. كيف نصل إلى استدعاء واحد فقط؟ عندما ندرج عنصرًا، فإما أنّ يوجد في العنقود الذي يضاف إليه هذا العنصر، عنصرٌ آخرُ سلفًا، وإما لا. فإذا تضمَّن العنقودُ عنصرًا آخر سلفًا بالفعل، فإن رقم العنقود موجودٌ فعليًّا في الملخص، وبذلك لا نحتاج إلى أن نجري ذلك الاستدعاء العودي. وإذا لم يتضمَّن العنقودُ عنصرًا آخرَ سلفًا، فإن العنصرَ المُدرَج يصبح العنصرَ الوحيد في العنقود، ولا نكون بحاجة للتكرار عوديًّا لإدراج عنصر في شجرة vEB خالية:

```
vEB-EMPTY-TREE-INSERT(V, x)
1  V.min = x
2  V.max = x
```

إذا كان لدينا هذا الإجراء، نجد فيما يلي شبه الرماز لإجراء vEB-TREE-INSERT(V, x)، الذي يفترض أن x ليس موجودًا بالفعل في المجموعة التي تمثلها شجرة vEB V:

```
vEB-TREE-INSERT(V, x)
 1  if V.min == NIL
 2      vEB-EMPTY-TREE-INSERT(V, x)
 3  else if x < V.min
 4      exchange x with V.min
 5      if V.u > 2
 6          if vEB-TREE-MINIMUM(V.cluster[high(x)]) == NIL
 7              vEB-TREE-INSERT(V.summary, high(x))
 8              vEB-EMPTY-TREE-INSERT(V.cluster[high(x)], low(x))
 9          else vEB-TREE-INSERT(V.cluster[high(x)], low(x))
10      if x > V.max
11          V.max = x
```

يعمل الإجراء كما يلي: يتحقّق السطرُ 1 من أنّ V هي شجرة vEB خالية، فإذا كانت كذلك، يعالج السطرُ 2 هذه الحالةَ السهلة. تفترض السطور 3–11 أن V غير خالية، ولذلك فإن عنصرًا ما سيُدرَج في أحد عناقيد V. ولكن قد لا يكون هذا العنصر هو العنصر x الذي مُرِّر إلى vEB-TREE-INSERT بالضرورة. إذا

كان $x < min$ عند اختباره في السطر 3، فإن x يجب أن يُصبح min الجديد. لكننا لا نريد أن نضيِّع min الأصلي، ولهذا لا بد من أن ندرجه في أحد عناقيد V. في هذه الحالة، يبادل السطر 4 بين x و min، بحيث ندرج min الأصلي في أحد عناقيد V.

لا ننفِّذ الأسطر 6-9 إلاَّ إذا لم تكن V شجرة vEB في الحالة الأساسية. يحدد السطر 6: هل العنقود الذي سيضاف فيه x خالٍ حاليًّا؟ فإذا كان كذلك، يُدرج السطر 7 رقمَ عنقود x في الملخص، ويعالج السطر 8 الحالة البسيطة لإدراج x في عنقود خالٍ. وإذا لم يكن عنقود x خاليًا حاليًّا، يُدرج السطر 9 العنصر x في عنقوده. في هذه الحالة، لسنا بحاجة إلى تحديث الملخص، لأن رقم عنقود x هو عنصر موجود فعليًّا في الملخص.

أخيرًا، يتولَّى السطران 10–11 تحديث max إذا كان $x > max$. لاحظ أنه إذا كانت V هي شجرة vEB في الحالة الأساسية، أي ليست خالية، فإن الأسطر 3–4 و 10–11 تحدِّث min و max بصورة صحيحة.

مرة أخرى، يمكننا بسهولة أن نرى كيف توصِّف المعادلةُ التكرارية (4.20) زمنَ التنفيذ. حسب نتيجة الاختبار في السطر 6، يجري تنفيذ الاستدعاء العودي في السطر 7 (التنفيذ على شجرة vEB مع حجم عالَم يساوي $\sqrt[\uparrow]{u}$)، أو الاستدعاء العودي في السطر 9 (التنفيذ على شجرة vEB مع حجم عالَم يساوي $\sqrt[\downarrow]{u}$). في كلتا الحالتين، فإن الاستدعاء العودي الوحيد هو على شجرة vEB مع حجم عالَم يساوي على الأكثر $\sqrt[\uparrow]{u}$. ولما كان الباقي من vEB-Tree-Insert يَستغرق زمنًا $O(1)$، فإننا نطبِّق المعادلة التكرارية (4.20)، وبذلك يكون زمن التنفيذ $O(\lg \lg u)$.

حذف عنصر

أخيرًا، نلقي نظرة على كيفية حذف عنصر من شجرة vEB. يفترض الإجراء vEB-Tree-Delete(V, x) أن x هو حاليًّا عنصرٌ موجودٌ في المجموعة التي تمثلها شجرة vEB V.

```
vEB-TREE-DELETE(V, x)
 1  if V.min == V.max
 2      V.min = NIL
 3      V.max = NIL
 4  elseif V.u == 2
 5      if x == 0
 6          V.min = 1
 7      else V.min = 0
 8      V.max = V.min
 9  else if x == V.min
10          first-cluster = vEB-TREE-MINIMUM(V.summary)
```

```
11          x = index(first-cluster,
                 vEB-TREE-MINIMUM(V.cluster[first-cluster]))
12          V.min = x
13      vEB-TREE-DELETE(V.cluster[high(x)], low(x))
14      if vEB-TREE-MINIMUM(V.cluster[high(x)]) == NIL
15          vEB-TREE-DELETE(V.summary, high(x))
16          if x == V.max
17              summary-max = vEB-TREE-MAXIMUM(V.summary)
18              if summary-max == NIL
19                  V.max = V.min
20              else V.max = index(summary-max,
                         vEB-TREE-MAXIMUM(V.cluster[summary-max]))
21      elseif x == V.max
22          V.max = index(high(x),
                 vEB-TREE-MAXIMUM(V.cluster[high(x)]))
```

يعمل إجراء vEB-TREE-DELETE كما يلي: إذا تضمنت شجرة vEB V عنصرًا واحدًا فقط، فإن حذفه يتم بالسهولة التي جرى فيها إدراج عنصر في شجرة vEB خالية: إذ يكفي أن نجعل قيمة min و max مساوية لـ NIL. تعالج الأسطر 1–3 هذه الحالة. وفي الحالة المعاكسة، يكون في V عنصران على الأقل. يختبر السطر 4 إذا كانت V شجرة vEB في الحالة الأساسية، فإذا كانت كذلك، تجعلُ الأسطرُ 5–8 قيمةَ min و max مساوية للعنصر الوحيد الباقي.

تفترض الأسطر 9–22 أن V تتضمَّن عنصرين أو أكثر وأن $u \geq 4$. علينا في هذه الحالة، أن نحذف عنصرًا من العنقود. ولكن قد لا يكون العنصر الذي نحذفه من العنقود هو x، لأنه في حال كان x يساوي min عندئذ، حالما نحذف x، يصبح عنصرٌ آخر من أحد عناقيد V هو min الجديد، وعلينا أن نحذف هذا العنصر الآخر من عنقوده. إذا أظهر الاختبار في السطر 9 أننا في هذه الحالة، عندها يضع السطر 10 في $first\text{-}cluster$ رقمَ العنقود الذي يتضمن العنصر الأصغري غير min، ويضع السطر 11 في x قيمة العنصر الأصغري في ذلك العنقود. يصبح هذا العنصر هو min الجديد في السطر 12، ولأننا وضعنا قيمته في x، فسيكون هو العنصر الذي سيحذف من عنقوده.

عندما نصل إلى السطر 13، نعلم أيضًا أنه يجب أن نحذف العنصر x من عنقوده، سواء أكان x هو القيمة الممررة أصلاً إلى vEB-TREE-DELETE أم كان x هو العنصر الذي أصبح العنصر الأصغري الجديد. يحذف السطر 13 العنصرَ x من عنقوده. ربما يكون هذا العنقود قد أصبح خاليًا، وهو ما يختبره السطر 14، فإذا كان كذلك، عندها نحتاج إلى حذف رقم عنقود x من الملخص، وهو ما يعالجه السطر 15. بعد تحديث الملخص، ربما نحتاج إلى تحديث max. يتفقَّد السطرُ 16 حَذْفَ العنصر الأعظمي في V، فإذا كان كذلك، يضعُ السطرُ 17 في $summary\text{-}max$ رقمَ العنقود غير الخالي ذي الرقم الأعلى. (يعمل

الاستدعاء (vEB-TREE-MAXIMUM($V.summary$ لأننا استدعينا vEB-TREE-DELETE عوديًّا على $V.summary$، ومن ثَمّ يكون $V.summary.max$ قد حُدِّث سلفًا بالصورة الملائمة.) إذا كانت جميع عناقيد V خالية، عندها يكون العنصر الوحيد الباقي في V هو min؛ يتحقق السطر 18 من هذه الحالة، ويحدِّث السطر 19 قيمة max بالصورة الملائمة. في الحالة المعاكسة، يضع السطر 20 في max العنصرَ الأعظم من العنقود غير الخالي ذي الرقم الأعلى. (إذا كان العنصر قد حُذِفَ من هذا العنقود، فنعتمد ثانية على أن الاستدعاء العودي في السطر 13 قد صحح سلفًا واصفة max من ذلك العنقود.)

في النهاية، علينا أن نعالج الحالة التي لا يصبح فيها عنقود x خاليًا نتيجة حذف x. رغم عدم ضرورة تحديثنا للملخص في هذه الحالة، إلا أنه قد يكون علينا تحديث max. يختبر السطر 21 هذه الحالة، وإذا كان علينا تحديث max، يقوم السطر 22 بذلك (مرة ثانية بالاعتماد على أن الاستدعاء العودي قد صحح max في العنقود).

نبين الآن أن vEB-TREE-DELETE يجري تنفيذه بزمن $O(\lg\lg u)$ في أسوأ الحالات. قد يبدو للوهلة الأولى، أن المعادلة التكرارية (4.20) لا تُطبَّق دائمًا، لأن استدعاءً واحدًا لـ vEB-TREE-DELETE قد ينشئ استدعاءين عوديين: أحدهما في السطر 13 والآخر في السطر 15. ومع أن الإجراء قد يُجري الاستدعاءين كليهما، فلنفكر في ما يحدث عندما يفعل ذلك. كي يحدث الاستدعاء العودي في السطر 15، يجب أن يبيّن الاختبار في السطر 14 أن عنقود x خالٍ. الحالة الوحيدة التي يمكن أن يكون فيها عنقود x خاليًا هي إذا كان x هو العنصر الوحيد في عنقوده عندما قمنا بالاستدعاء العودي في السطر 13. لكن إذا كان x هو العنصر الوحيد في عنقوده، يكون ذلك الاستدعاء العودي قد تطلب زمنًا $O(1)$، لأنه نفذ الأسطر 1–3 فقط. بذلك، يكون لدينا احتمالان يستثني أحدهما الآخر:

- يستغرق الاستدعاء العودي في السطر 13 زمنًا ثابتًا
- لم يحصُل الاستدعاء العودي في السطر 15.

في كلتا الحالتين، تُوصِّفُ المعادلةُ التكرارية (4.20) زمن تنفيذ vEB-TREE-DELETE، وبذلك فإن زمن تنفيذه في أسوأ الحالات هو $O(\lg\lg u)$.

تمارين

1-3.20

عدّل أشجار vEB لتدعم المفاتيح المكررة.

2-3.20

عدّل أشجار vEB لتدعم المفاتيح التي لها معطيات تابعة مرتبطة.

3-3.20

اكتب شبه رماز لإجراءٍ ينشئ شجرة van Emde Boas خالية.

4-3.20

ماذا يحدث لو استدعيتَ vEB-TREE-INSERT على عنصرٍ موجود سابقًا في شجرة vEB؟ وماذا يحدث لو استدعيتَ vEB-TREE-DELETE على عنصر غير موجود في شجرة vEB؟ فسِّر لماذا يسلك هذان الإجراءان هذا السلوك. بيِّن كيف نعدِّل أشجار vEB وعملياتها بحيث نتحقَّق في زمن ثابت من وجود عنصرٍ ما فيها.

5-3.20

افترض أننا أنشأنا، بدلاً من $\sqrt[\uparrow]{u}$ عنقودًا حجمُ عالَمِ كلٍّ منها $\sqrt[\downarrow]{u}$، أشجارَ vEB ليكون فيها $u^{1/k}$ عنقودًا، حجمُ عالَمِ كلٍّ منها $u^{1-1/k}$، حيث k ثابت أكبر من الواحد. إذا كان علينا أن نعدِّل العمليات بصورة ملائمة، ماذا سيكون زمن تنفيذها؟ افترض بهدف التحليل، أن $u^{1/k}$ و $u^{1-1/k}$ أعداد صحيحة دومًا.

6-3.20

إن إنشاء شجرة vEB حجمُ عالَمها u يتطلب زمنًا $\Theta(u)$. افترض أننا نرغب بحساب تفصيلي لهذا الزمن. ما هو أصغر عدد عمليات n بحيث تستغرق كل عملية في شجرة vEB ما زمنًا مخمَّداً $O(\lg \lg u)$؟

مسائل

1-20 متطلبات الحجم لأشجار *van Emde Boas*

تسبر هذه المسألة متطلبات الحجم لأشجار van Emde Boas، وتقترح طريقة لتعديل بنية المعطيات لجعل متطلبها من الحجم يعتمد على عدد العناصر n المخزنة حاليًا في الشجرة، وليس على حجم العالَمَ u. نفترض للتبسيط أن $\sqrt{u}$ دائمًا عدد صحيح.

أ. فسِّر لماذا توصِّف المعادلة التكرارية التالية المتطلب الحجمي $P(u)$ لشجرة van Emde Boas حجمُ عالَمها u:

$$P(u) = (\sqrt{u} + 1)P(\sqrt{u}) + \Theta(\sqrt{u}) . \quad (5.20)$$

ب. أثبت أن للمعادلة التكرارية (5.20) الحلَّ $P(u) = O(u)$.

لتقليص المتطلبات الحجمية، نعرِّف ***شجرةً ذات حجم مقلَّص reduced-space van Emde Boas***، أو ***شجرة RS-vEB***، بأنها شجرة vEB V مع التغييرات التالية:

- الواصفة $V.cluster$ بدلاً من أن تكون مخزنة كصفيفة بسيطة من المؤشرات على أشجار vEB مع حجم عالَمٍ $\sqrt{u}$، هي الآن جدول التلبيد hash table (انظر الفصل 11) مخزَّن كجدول ديناميكي (انظر المقطع 4.17). يُخزِّنُ جدولُ التلبيد، كما في نسخة الصفيفة من $V.cluster$، مؤشراتٍ على أشجار RS-vEB بحجم عالَمٍ $\sqrt{u}$. لإيجاد العنقود ذي الترتيب i، نبحث عن المفتاح i في جدول التلبيد، وبذلك يمكننا إيجاد العنقود ذي الترتيب i ببحثٍ واحدٍ في جدول التلبيد.
- يخزِّن جدول التلبيد مؤشراتٍ إلى العناقيد غير الخالية فقط. يعيد البحث عن عنقودٍ خالٍ في جدول التلبيد القيمةَ NIL، مبينًا أن العنقود خال.
- تأخذ الواصفة $V.summay$ القيمةَ NIL إذا كانت جميع العناقيد خالية. في الحالة المعاكسة، يشير $V.summary$ إلى شجرة RS-vEB بحجم عالَمٍ $\sqrt{u}$.

لما كان جدول التلبيد يُنجَز باستخدام جدول ديناميكي، فإن الحجم الذي يتطلبه متناسبٌ طردًا مع عدد العناقيد غير الخالية.

عندما نحتاج إلى إدراج عنصر في شجرة RS-vEB خالية، ننشئ شجرة RS-vEB باستدعاء الإجراء التالي، حيث الموسط u هو حجم العالَم لشجرة RS-vEB.

```
CREATE-NEW-RS-vEB-TREE(u)
1  allocate a new vEB tree V
2  V.u = u
3  V.min = NIL
4  V.max = NIL
5  V.summary = NIL
6  create V.cluster as an empty dynamic hash table
7  return V
```

ت. عدِّل إجراء vEB-TREE-INSERT لتوليد شبه رماز لإجراء RS-vEB-TREE-INSERT(V,x) الذي يدرج x في شجرة V من نمط RS-vEB، مستدعيًا CREATE-NEW-RS-vEB-TREE عند اللزوم.

ث. عدِّل إجراء vEB-TREE-SUCCESSOR لتوليد شبه رماز لإجراء RS-vEB-TREE-SUCCESSOR(V,x) الذي يعيد العنصر اللاحق لـ x في شجرة V من نمط RS-vEB، أو NIL إذا لم يكن لـ x عنصر لاحق في V.

ج. أثبت – بافتراض أن التلبيد بسيط ومنتظم – أن إجراءي RS-vEB-TREE-INSERT و RS-vEB-TREE-SUCCESSOR ينفذان بزمن متوقع محدَّد $O(\lg \lg u)$.

ح. بافتراض أن العناصر لا تُحذَف أبدًا من شجرة vEB، أثبت أن المتطلب الحجمي لبنية شجرة RS-vEB هو $O(n)$، حيث n هو عدد العناصر المخزنة حاليًا في شجرة RS-vEB.

خ. الأشجار RS-vEB لها مزية أخرى على أشجار vEB: حيث يتطلب إنشاؤها زمنًا أقل. كم يستغرق إنشاء شجرة RS-vEB فارغة؟

20-2 بنية *y-fast tries*

تبحث هذه المسألة في بنية "y-fast tries" التي اقترحها D. Willard، والتي تُنَفِّذ، مثل أشجار van Emde Boas، كلاً من العمليات MEMBER و MINIMUM و MAXIMUM و PREDESSOR و SUCCESSOR على عناصر مأخوذة من عالَمٍ حجمُه u بزمن $O(\lg \lg u)$ في أسوأ الحالات. تستغرق عمليتا INSERT و DELETE زمنًا مُخمَّدًا $O(\lg \lg u)$. تَستخدم y-fast tries، حجمًا $O(n)$ فقط لتخزين n عنصرًا مثل أشجار reduced-space van Emde Boas (انظر المسألة 20–1). يعتمد تصميم y-fast tries على التلبيد المثالي perfect hash (انظر المقطع 5.11).

في بنية مبدئية، افترض أننا ننشئ جدولَ تلبيد مثالي لا يتضمن جميعَ العناصر في المجموعة الديناميكية فقط، بل يتضمَّن كلَّ سابقة prefix من التمثيل الثنائي لكل عنصر في المجموعة. فمثلاً، إذا كان $u = 16$، فإن $\lg u = 4$، والعنصر $x = 13$ موجود في المجموعة. ولما كان التمثيل الثنائي لـ 13 هو 1101، فإن جدول التلبيد المثالي سيتضمن المتواليات 1 و 11 و 110 و 1101. ننشئ إضافةً إلى جدول التلبيد قائمةً مضاعفة الترابط من العناصر الموجودة حاليًّا في المجموعة، بترتيب متزايد.

أ. ما الحجم الذي تتطلبه هذه البنية؟

ب. بيّن كيفية إنجاز عمليتَي MINIMUM و MAXIMUM بزمن $O(1)$؛ وعمليات MEMBER و PREDECESSOR و SUCCESSOR بزمن $O(\lg \lg u)$؛ وعمليتَي INSERT و DELETE بزمن $O(\lg u)$.

لتقليص المتطلب الحجمي إلى $O(n)$، نقوم بالتعديلات التالية على بنية المعطيات:

- نُعَنْقِدُ العناصرَ التي عددها n ضمن $n/\lg u$ مجموعة ذات حجم $\lg u$. (افترض الآن أن $\lg u$ تَقْسِمُ n.) تتألف المجموعة الأولى من $\lg u$ أصغر عنصر في المجموعة، وتتألف المجموعة الثانية من $\lg u$ أصغر عنصر مما تبقى، وهكذا.
- نعيِّن قيمةً "ممثلة" عن كل مجموعة. تكون قيمة ممثل المجموعة ذات الترتيب i مساوية لقيمة أكبر عنصر في المجموعة i على الأقل، وهي كذلك أصغر من جميع عناصر المجموعة $(i + 1)$. (يمكن أن يكون ممثل آخر مجموعةٍ أكبرَ عنصرٍ ممكن u منقوصًا منه 1.) لاحظ أن الممثل يمكن أن يكون قيمة غير موجودة حاليًّا في المجموعة.
- نُخزِّن الـ $\lg u$ عنصرًا من كل مجموعة في شجرةِ بحثٍ ثنائيةٍ متوازنة، مثلَ شجرة حمراء–سوداء. يشير كلُّ ممثلٍ إلى شجرةِ البحث الثنائية المتوازنة الخاصة بمجموعتها، وتشير كل شجرة بحث ثنائية متوازنة إلى ممثل مجموعتها.

• يُخزِّن جدولُ التلبيد المثالي الممثِّلين فقط، كما يخزنون أيضًا في قائمة مضاعفة الارتباط بترتيب متزايد.

نسمي هذه البنية ***y-fast trie***.

ت. بيِّن أن y-fast trie تتطلب حجمًا $O(n)$ فقط لتخزين n عنصرًا.

ث. بيِّن كيفية إنجاز عمليتَي MINIMUM و MAXIMUM بزمن $O(\lg \lg u)$ باستخدام y-fast trie.

ج. بيِّن كيفية إنجاز عملية MEMBER بزمن $O(\lg \lg u)$.

ح. بيِّن كيفية إنجاز عمليتَي PREDECESSOR و SUCCESSOR بزمن $O(\lg \lg u)$.

خ. اشرح لماذا تستغرق عمليتا INSERT و DELETE زمنًا $\Omega(\lg \lg u)$.

د. بيِّن كيفية إرخاء relax متطلب أن يكون عددُ عناصر كلِّ مجموعة من y-fast-trie هو $\lg u$ عنصرًا تمامًا ليسمح بتنفيذ INSERT و DELETE بزمن محدَّد $O(\lg \lg u)$ دون التأثير في الأزمنة المقاربة لتنفيذ العمليات الأخرى.

ملاحظات الفصل

سُمِّيت بنية المعطيات في هذا الفصل باسم P. van Emde Boas، الذي وَصَفَ صيغةً أولية للفكرة في عام 1975 [339]. فصلَّت المقالات اللاحقة لـ van Emde Boas [340] و van Emde Boas و Kass و Zijlstra [341] الفكرةَ والعرض. وسَّع Mehlhorn و Näher [252] لاحقًا هذه الأفكار لتطبَّق على حجوم عالَم أولية. يتضمن كتاب Mehlhorn [249] معالجةً لأشجار van Emde Boas تختلف قليلاً عما هو موجود في هذا الفصل.

قام Dementiev وزملاؤه [84] باستخدام الأفكار المتعلقة بأشجار van Emde Boas لتطوير شجرة بحث من ثلاثة مستويات غير عودية تُنَفَّذ بصورة أسرع من أشجار van Emde Boas في تجاربهم الخاصة.

صمَّم wang and Lin [347] إصدارًا عتاديًّا أنبوبيًّا hardware-pipelined من أشجار van Emde Boas، يحقِّق زمنًا محَدَّدًا ثابتًا لكل عملية ويَستخدم $O(\lg \lg u)$ مرحلةً في الأنبوب pipeline.

يبيِّن حدٌّ أدنى اكتشفه Pătraşcu و Thorup [273, 274] للعثور على العنصر السابق predecessor، أن أشجار van Emde Boas مثلى لهذه العملية، ولو كانت العشوائية مسموحة.

21 بنى المعطيات للمجموعات المنفصلة

تتطلب بعضُ التطبيقات تجميعَ n عنصرًا متمايزًا في تجمُّعٍ من المجموعات المنفصلة. تحتاج هذه التطبيقات غالبًا إلى إجراء عمليتين هما: إيجاد المجموعة الوحيدة التي ينتمي إليها عنصر ما، وتوحيد مجموعتين. يستكشف هذا الفصل طرق الحفاظ على بنية معطيات تدعم هذه العمليات.

يصف المقطع 1.21 العمليات التي تدعمها بنية معطيات مجموعات منفصلة، ويقدم تطبيقًا بسيطًا. وفي المقطع 2.21 ندرس تنجيزًا بسيطًا للمجموعات المنفصلة باستخدام اللائحة المترابطة. يعطي المقطع 3.21 تمثيلاً أكثر فعاليةً باستخدام الأشجار ذات الجذور. إنَّ زمن التنفيذ باستخدام التمثيل الشجري هو نظريًّا فوق خطي، لكنه خطيٌّ في جميع الأهداف العملية. يعرِّف المقطع 4.21 دالةً سريعة النمو، ودالتها المعاكسة البطيئة النمو التي تظهر في زمن تنفيذ العمليات على التنجيز الشجري وتناقشهما، ثم يُثبت – بالتحليل المخمَّد – حدًّا أعلى لزمن التنفيذ الذي هو بالكاد فوق خطي.

1.21 عمليات المجموعات المنفصلة

تحتوي ***بنية معطيات المجموعات المنفصلة disjoint-set data structure*** على تجمُّع $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ من المجموعات الديناميكية المنفصلة. تُحدَّد كلُّ مجموعةٍ ***بممثِّل representative*** هو أحد عناصرها. في بعض التطبيقات، لا يُعدُّ العنصر الذي استُخدم ممثِّلاً أمرًا مهمًّا؛ بل المهم أننا إذا طلبنا ممثِّلَ مجموعةٍ ديناميكية مرتين دون تعديل المجموعة بينهما هو أن نحصل على الإجابة نفسها. قد تتطلب تطبيقات أخرى وجود قاعدة محدَّدة سلفًا لاختيار الممثل، كاختيار العنصر الأصغر في المجموعة (طبعًا بافتراض إمكان ترتيب العناصر).

وكما في تنجيزات المجموعات الديناميكية الأخرى التي درسناها، نُمثِّل كل عنصر في المجموعة بغرض object. فإذا كان x غرضًا، فإننا نرغب بدعم العمليات التالية:

MAKE-SET(x) إنشاء مجموعة جديدة فيها عنصر وحيد هو x (ومن ثَمّ فهو الممثل). ولمّا كانت المجموعات منفصلة، فإننا نطلب ألا يكون x موجودًا سلفًا في مجموعة أخرى.

$\text{UNION}(x, y)$ الاجتماع الذي يَجمَع المجموعتين الديناميكيتين اللتين تحتويان على x و y (S_x و S_y مثلاً) في مجموعة جديدة هي اجتماعهما. نفترض أن المجموعتين منفصلتان قبل العملية. إن ممثِّل المجموعة الناتجة هو أي عنصر من $S_x \cup S_y$، علمًا بأن العديد من التنجيزات يختار ممثِّلَ S_x أو S_y ليكون ممثِّلاً للمجموعة الجديدة. ولمّا كان المطلوب هو أن تكون المجموعات في التجمُّع منفصلة، فإننا ندمِّر destroy المجموعتين S_x و S_y مفاهيميًّا ونحذفهما من التجمُّع $\mathcal{S}$. وغالبًا ما نقوم عمليًّا بامتصاص absorb عناصر إحدى المجموعتين ضمن المجموعة الأخرى.

$\text{FIND-SET}(x)$ إيجاد مجموعة تعيد مؤشرًا إلى ممثل المجموعة (الوحيدة) التي تحتوي على x.

سنحلل في هذا الفصل أزمنة تنفيذ العمليات على بنية معطيات المجموعات المنفصلة بدلالة وسيطين هما: n عدد عمليات MAKE-SET ، و m العدد الكلي للعمليات MAKE-SET و UNION و FIND-SET . ولمّا كانت المجموعات منفصلة، فإنَّ كلَّ عملية UNION تُقلِّص عدد المجموعات بمقدار واحد. ولذلك، وبعد $n-1$ عملية تبقى لدينا مجموعة واحدة فقط. ومن ثم، فإن عدد عمليات UNION هو على الأكثر $n-1$. لاحظ أيضًا أنَّه لمّا كانت عمليات MAKE-SET متضمَّنة في العدد الكلي للعمليات m، فإن $m \geq n$. نفترض أنَّ عمليات MAKE-SET هي العمليات الـ n التي المنجزَة أولاً.

تطبيق على بنى معطيات المجموعات المنفصلة

يَظهَر أحد التطبيقات المتعددة لبنى معطيات المجموعات المنفصلة في تحديد المكونات المرتبطة في بيان غير موجَّه (انظر المقطع ب.4). فمثلاً، يُظهر الشكل 1.21(أ) بيانًا مؤلفًا من أربعة مكونات.

يَستخدم الإجراء CONNECTED-COMPONENTS التالي عمليات المجموعات المنفصلة لحساب المكونات المرتبطة في بيان. بمجرد تنفيذ CONNECTED-COMPONENTS لمعالجة أولية للبيان، يقوم الإجراء SAME-COMPONENT بالإجابة عن الاستعلامات المتعلقة بكون عقدتين تنتميان إلى المكون نفسه.[1] (نَرمز في شبه الرماز إلى مجموعة عقد بيان G بـ $G.V$، وإلى مجموعة الوصلات بـ $G.E$.)

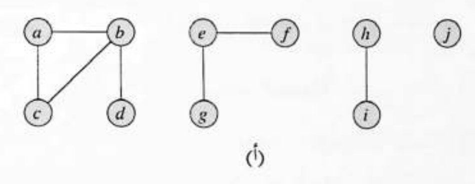
```
CONNECTED-COMPONENTS(G)
1  for each vertex v ∈ G.V
2       MAKE-SET(v)
3  for each edge (u, v) ∈ G.E
```

[1] عندما تكون وصلات البيان سكونية (أي لا تتغير عبر الزمن)، يمكننا حساب المكونات المرتبطة بسرعة أكبر باستخدام البحث عمقًا–أولاً (التمرين 3.22-12). مع ذلك، تضاف الوصلات ديناميكيًّا أحيانًا ونحتاج إلى الحفاظ على المكونات المرتبطة مع إضافة كل وصلة. في هذه الحالة، يمكن أن يكون التنجيز الموجود هنا أكثر فعاليةً من تنفيذ بحث جديد عمقًا–أولاً لكل وصلة جديدة.

```
4       if FIND-SET(u) ≠ FIND-SET(v)
5           UNION(u, v)
```

```
SAME-COMPONENT(u, v)
1  if FIND-SET(u) == FIND-SET(v)
2       return TRUE
3  else return FALSE
```

يضع الإجراء CONNECTED-COMPONENTS في البداية كلَّ عقدة v في مجموعة خاصة بها. ثم يوحّد unite المجموعتين اللتين تحتويان u و v، وذلك لكل وصلة (u,v). يُطلب في التمرين 21.1-2، بعد معالجة جميع الوصلات، إثبات أن عقدتَيْن تقعان في المكون نفسه إذا وفقط إذا كان الغرضان المقابلان لهما في المجموعة نفسها. ومن ثم، يَحسب الإجراء CONNECTED-COMPONENTS المجموعات بطريقةٍ تجعل الإجراء SAME-COMPONENT يحدِّد وجود عقدتَين في المكون المرتبط نفسه. يبين الشكل 21.1(ب) كيف يَحسب CONNECTED-COMPONENTS المجموعات المنفصلة.



(أ)

تجمُّع المجموعات المنفصلة										الوصلة المعالجة
$\{j\}$	$\{i\}$	$\{h\}$	$\{g\}$	$\{f\}$	$\{e\}$	$\{d\}$	$\{c\}$	$\{b\}$	$\{a\}$	المجموعات الابتدائية
$\{j\}$	$\{i\}$	$\{h\}$	$\{g\}$	$\{f\}$	$\{e\}$		$\{c\}$	$\{b,d\}$	$\{a\}$	(b,d)
$\{j\}$	$\{i\}$	$\{h\}$		$\{f\}$	$\{e,g\}$		$\{c\}$	$\{b,d\}$	$\{a\}$	(e,g)
$\{j\}$	$\{i\}$	$\{h\}$		$\{f\}$	$\{e,g\}$			$\{b,d\}$	$\{a,c\}$	(e,c)
$\{j\}$		$\{h,i\}$		$\{f\}$	$\{e,g\}$			$\{b,d\}$	$\{a,c\}$	(h,i)
$\{j\}$		$\{h,i\}$		$\{f\}$	$\{e,g\}$				$\{a,b,c,d\}$	(a,b)
$\{j\}$		$\{h,i\}$			$\{e,f,g\}$				$\{a,b,c,d\}$	(e,f)
$\{j\}$		$\{h,i\}$			$\{e,f,g\}$				$\{a,b,c,d\}$	(b,c)

(ب)

الشكل 21.1 (أ) بيان مؤلف من أربعة مكونات $\{a,b,c,d\}$ و $\{e,f,g\}$ و $\{h,i\}$ و $\{j\}$. **(ب)** تجمُّع المجموعات المنفصلة بعد معالجة كل وصلة.

في تنجيزٍ فعليٍّ لخوارزمية المكونات المرتبطة هذه، سيحتاج كلٌّ من تمثيلي البيان وبنية معطيات المجموعات المنفصلة إلى أن يشير كلٌّ منهما إلى الآخر. أي إنَّ كلَّ غرضٍ يمثِّل عقدةً سيحتوي على مؤشرٍ pointer إلى غرضِ المجموعة المنفصلة المقابل، والعكس بالعكس. تعتمد هذه التفاصيل البرمجية على لغة التنجيز، ولن نعالجها أكثر من ذلك هنا.

تمارين

1-1.21

افترض أنَّه جرى تنفيذ CONNECTED-COMPONENTS على البيان غير الموجَّه $G = (V, E)$، حيث $V = \{a, b, c, d, e, f, g, h, i, j, k\}$ وتجري معالجة الوصلات E بالترتيب التالي: $(d, i), (f, k), (g, i), (b, g), (a, h), (i, j), (d, k), (b, j), (d, f), (g, j), (a, e)$. اسرد العقد في كلِّ مكوِّنٍ مرتبط بعد كل تكرار للأسطر 3-5.

2-1.21

بيِّن أنه بعد معالجة جميع الوصلات في CONNECTED-COMPONENTS، تكون عقدتان في المكوِّن نفسه إذا وفقط إذا كانا في المجموعة نفسها.

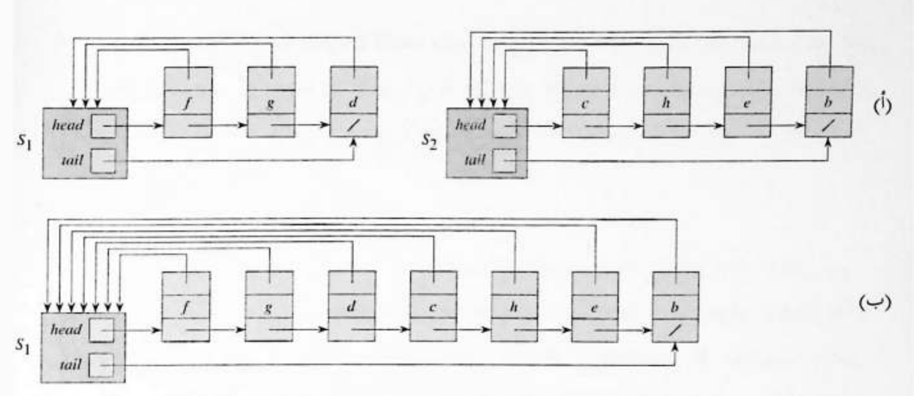
3-1.21

ما هو عدد المرات التي يُستدعى فيها FIND-SET خلال تنفيذ CONNECTED-COMPONENTS على بيان غير موجَّه $G = (V, E)$ ذي k مكوِّنًا مرتبطًا؟ وما هو عدد المرات التي يُستدعى فيها UNION؟ عبِّر عن أجوبتك بدلالة $|V|$ و $|E|$ و k.

2.21 تمثيل المجموعات المنفصلة بلائحة مترابطة

يُظهر الشكل 2.21(أ) طريقةً بسيطة لتنجيز بنية معطيات مجموعات منفصلة: تُمثَّل فيها كلُّ مجموعة باللائحة المترابطة الخاصة بها. يحتوي الغرض في كلِّ مجموعةٍ على واصفات *رأس head* يؤشر على الغرض الأول في اللائحة، و*ذيل tail* يؤشر على الغرض الأخير. يحتوي كلُّ غرضٍ من اللائحة المترابطة على عنصر من المجموعة، ومؤشرٍ إلى الغرض التالي في اللائحة، ومؤشرٍ يرجع إلى غرض المجموعة. يمكن أن تَظهَر الأغراض في كل لائحة مترابطة بأي ترتيب كان. إن ممثِّلَ المجموعة هو عنصرُ المجموعة في الغرض الأول في اللائحة.

من السهل تنفيذ MAKE-SET وFIND-SET كليهما بزمن $O(1)$ في تمثيل اللائحة المترابطة. ولتنفيذ MAKE-SET(x) ننشئ لائحة مترابطة جديدة غرضها الوحيد هو x. ففي حالة FIND-SET(x) نعيد المؤشر من x إلى غرض المجموعة الخاص بها، ثم نعيد العنصر في الغرض الذي يشير إليه *head*. فمثلاً، في الشكل 2.21(أ)، يعيد استدعاء FIND-SET(g) القيمة f.



الشكل 2.21 (أ) تمثيل مجموعتين باستخدام اللائحة المترابطة. تحتوي المجموعة S_1 على العناصر d و f و g حيث f هو الممثل. وتحتوي المجموعة S_2 على العناصر b و c و e و h حيث c هو الممثل. يحتوي كل غرض في اللائحة على عنصر من المجموعة، ومؤشر إلى الغرض التالي في اللائحة، ومؤشر يرجع إلى غرض المجموعة. ويحتوي كل غرض مجموعة على مؤشر *head* ومؤشر *tail* على الغرض الأول والغرض الأخير على التتالي. **(ب)** نتيجة UNION(g,e) الذي يُلحق اللائحةَ المترابطةَ التي تحوي e باللائحة المترابطة التي تحوي g. ممثل اللائحة الناتجة هو f. ويجري تدمير، S_2، غرض المجموعة الخاصة بلائحة e.

تنجيز بسيط للإجراء UNION

يستغرق التنجيز الأبسط لعملية UNION باستخدام تمثيل المجموعة بلائحة مترابطة وقتًا أكثر بكثير من MAKE-SET و FIND-SET. وكما يُظهر الشكل 2.21(ب) فإننا ننفِّذ UNION(x,y) بإلحاق لائحة y بنهاية لائحة x. ويصبح ممثِّلُ لائحة x هو الممثِّلَ للمجموعة الناتجة. نستخدم المؤشر *tail* في لائحة x لإيجاد مكان إلحاق باللائحة y بسرعة. ولما كانت جميع عناصر لائحة y ستنضم إلى لائحة x، فيمكننا تدمير غرض المجموعة للائحة y. ولكنْ علينا تحديث المؤشر إلى غرض المجموعة لكل غرض كان في الأصل في لائحة y، وهذا يستغرق زمنًا خطيًّا نسبةً إلى طول اللائحة y. ففي الشكل 2.21 مثلاً، تسبب عملية UNION(g,e) تعديل المؤشرات في الأغراض b و c و e و h.

في الحقيقة، ليس من السهل إنشاء متتالية من m عمليةً على n غرضًا تتطلب زمنًا $\Theta(n^2)$. افترض أن لدينا الأغراض $x_1, x_2, \ldots, x_n$. ننفِّذ متتالية من n عملية MAKE-SET تتبعها $n-1$ عملية UNION المبيَّنة في الشكل 3.21، بحيث يكون $m = 2n-1$. يستغرق إجراء n عملية MAKE-SET زمنًا رتبته $\Theta(n)$. ولمّا كانت عملية UNION ذات الرقم i تحدِّث i غرضًا، فإن العددَ الكلي للأغراض المحدَّثة في $n-1$ عملية UNION هو:

$$\sum_{i=1}^{n-1} i = \Theta(n^2) .$$

العملية	عدد الأغراض المحدثة
$\text{MAKE-SET}(x_1)$	1
$\text{MAKE-SET}(x_2)$	1
⋮	
$\text{MAKE-SET}(x_n)$	1
$\text{UNION}(x_2, x_1)$	1
$\text{UNION}(x_3, x_2)$	2
$\text{UNION}(x_4, x_3)$	3
⋮	
$\text{UNION}(x_n, x_{n-1})$	$n-1$

الشكل 3.21 متتالية من $2n-1$ عملية على n غرضًا تستغرق زمنًا $\Theta(n^2)$، أو زمنًا $\Theta(n)$ لكل عملية وسطيًا، باستخدام تمثيل المجموعات بلائحة مترابطة والتنجيز البسيط لـ UNION.

ويكون العدد الكلي للعمليات هو $2n-1$ عملية، وبذلك تستغرق كلُّ عملية زمنًا وسطيًا $\Theta(n)$. أي إنَّ الزمن المخمَّد لعمليةٍ ما هو $\Theta(n)$.

كسبية اجتماع مثقَّل

يتطلب التنجيز السابق للإجراء UNION في أسوأ الحالات زمنًا $\Theta(n)$ لكل استدعاء وسطيًا، لأننا قد نُلحق لائحةً طويلةً بلائحةٍ قصيرة؛ يجب أن نحدِّث المؤشر إلى الممثل، لكلِّ عنصرٍ في اللائحة الطويلة. افترض بدلاً من ذلك، أنَّ كلَّ لائحة تتضمن أيضًا طول اللائحة (الذي يمكن الحفاظ عليه بسهولة) وأننا نُلحق اللائحة الأقصر باللائحة الأطول دومًا مع كسر الروابط اعتباطيًا. ***بكسبية الاجتماع المثقَّل weighted-union heuristic*** البسيطة هذه يمكن أن تظل عملية UNION بحاجة إلى زمن $\Omega(n)$ إذا كانت المجموعتان تحتويان $\Omega(n)$ عنصرًا. ومع ذلك، وكما تبيِّن المبرهنة التالية، فإنَّ متتالية مؤلفة من m عملية MAKE-SET و UNION و FIND-SET منها n عملية MAKE-SET تستغرق زمنًا $O(m + n \lg n)$.

مبرهنة 1.21

باستخدام تمثيل اللائحة المترابطة للمجموعات المنفصلة وكسبية الاجتماع المثقَّل، تستغرق متتالية مؤلفة من m عملية MAKE-SET و UNION و FIND-SET منها n عملية MAKE-SET زمنًا $O(m + n \lg n)$.

البرهان لمّا كانت عملية UNION تَجمع مجموعتين منفصلتين، فإننا نُنجز $n-1$ عملية UNION على الأكثر. نقوم الآن بحد الزمن الكلي الذي تستغرقه عمليات UNION هذه. نبدأ بتحديد حدٍّ أعلى لعدد المرات التي يجري فيها تحديث مؤشرات الأغراض إلى أغراض المجموعات الخاصة بها. ليكن لدينا غرض محدد x. نعلم أنَّه في كلِّ مرة يُحدَّث فيها مؤشر x، يجب أن يكون x قد بدأ في المجموعة الصغرى. في المرة الأولى

التي جرى فيها تحديث مؤشر x، يجب أن يكون في المجموعة الناتجة عنصران على الأقل. وبالمثل عند تحديث مؤشر x في المرة التالية، يجب أن يكون في المجموعة الناتجة أربعة عناصر على الأقل. وبالمتابعة على هذا المنوال، نلاحظ أنَّه بعد تحديث مؤشر x $\lceil \lg k \rceil$ مرةً، يجب أن يكون في المجموعة الناتجة k عنصرًا على الأقل، لأيِّ $k \le n$. ولما كانت المجموعة الكبرى تحوي n عنصرًا على الأكثر، فإنَّ كلَّ مؤشر إلى غرض يكون قد جرى تحديثه $\lceil \lg n \rceil$ مرةً على الأكثر في كل عمليات UNION. وهكذا يكون الزمن الكلي المصروف لتحديث مؤشرات الأغراض في كل عمليات UNION هو $O(n \lg n)$. يجب أن نحتسب أيضًا تحديث المؤشرات *tail* وأطوال اللائحة التي تستغرق زمنًا $\Theta(1)$ لكل عملية UNION. وبذلك يكون الزمن الكلي المصروف في تحديث n غرضًا هو $O(n \lg n)$.

يمكن استنتاج الزمن الكامل لمتتالية العمليات m بسهولة؛ فكل عملية MAKE-SET و FIND-SET تستغرق زمنًا $O(1)$، وعدد العمليات هو $O(m)$. إذن الزمن الكلي للمتتالية كاملةً هو $O(m + n \lg n)$. ■

تمارين

1-2.21

اكتب شبه رماز لكل من MAKE-SET و FIND-SET و UNION باستخدام تمثيل اللائحة المترابطة وكسبية الاجتماع المثقَّل. تأكد أنك حدَّدت الواصفات التي افترضتها لأغراض المجموعات وأغراض اللوائح.

2-2.21

بيِّن بنية المعطيات الناتجة عن عمليات FIND-SET والإجابات المعادة في البرنامج التالي. استخدم تمثيل اللائحة المترابطة مع كسبية الاجتماع المثقَّل.

```
1  for i = 1 to 16
2      MAKE-SET(x_i)
3  for i = 1 to 15 by 2
4      UNION(x_i, x_{i+1})
5  for i = 1 to 13 by 4
6      UNION(x_i, x_{i+2})
7  UNION(x_1, x_5)
8  UNION(x_11, x_13)
9  UNION(x_1, x_10)
10 FIND-SET(x_2)
11 FIND-SET(x_9)
```

افترض أنَّه إذا كان للمجموعتين x_i و x_j الحجم نفسه، فإنَّ العملية $\text{UNION}(x_i, x_j)$ تُلحق لائحةَ x_j بلائحة x_i.

3-2.21

عدِّل البرهان التجميعي للمبرهنة 1.21 للحصول على الحدود الزمنية المخمَّدة $O(1)$ لكلٍّ من MAKE-SET

و FIND-SET والحد $O(\lg n)$ لـ UNION باستخدام تمثيل اللائحة المترابطة وكسبية الاجتماع المثقَّل.

4-2.21

أعطِ حدًّا مقاربًا مُحْكَمًا لزمن تنفيذ متتالية العمليات في الشكل 3.21 بافتراض تمثيل اللائحة المترابطة وكسبية الاجتماع المثقَّل.

5-2.21

يظن الأستاذ Gompers أنَّه قد يكون من الممكن الاحتفاظ بمؤشر واحد فقط في كل غرض مجموعة، بدلاً من اثنين (*head* و *tail*)، مع الاحتفاظ بمؤشرين في كل عنصر من عناصر اللائحة. بيِّن أنَّ ظن الأستاذ مبنيٌّ على أسس جيدة، عن طريق وصف كيفية تمثيل مجموعة بلائحة مترابطة بحيث يكون للعمليات زمن تنفيذ العمليات الموصَّفة في هذا المقطع نفسه. صِف أيضًا كيفية عمل العمليات. يجب أن يسمح أسلوبك باستخدام كسبية الاجتماع المثقَّل، بالتأثير الموصَّف في هذا المقطع نفسه. (*تلميح*: استخدم ذيل اللائحة المترابطة باعتباره ممثلاً لمجموعتها.)

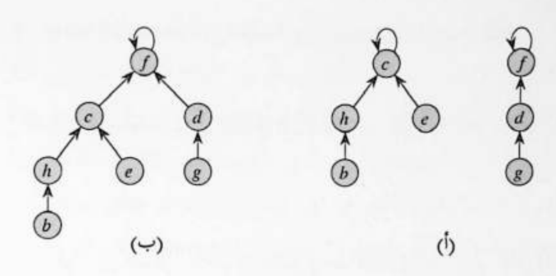
6-2.21

اقترح تغييرًا بسيطًا للإجراء UNION في تمثيل اللائحة المترابطة يلغي الحاجة إلى الاحتفاظ بالمؤشر ***tail*** إلى الغرض الأخير في اللائحة. يجب ألاَّ يتغير زمن التنفيذ المقارب للإجراء UNION سواءً استُخدمت كسبية الاجتماع المثقَّل أم لم تُستخدم. (*تلميح*: بدلاً من إلحاق لائحة بأخرى، صِلْهُما معًا.)

3.21 غابات المجموعات المنفصلة

في تنجيز أسرع للمجموعات المنفصلة، نمثل المجموعات بأشجار ذات جذور بحيث تحتوي كل عقدة على عنصر، وتمثِّل كلُّ شجرة مجموعة. في ***غابة المجموعات المنفصلة disjoint-set forest*** الموضحة في الشكل 4.21(أ)، يشير كل عنصر إلى أبيه فقط. يتضمن جذر كل شجرة ممثل المجموعة وهو أبٌ لنفسه. وكما سنرى لاحقًا، ومع أن الخوارزميات المباشرة التي تَستخدم هذا التمثيل ليست أسرع من تلك التي تَستخدم تمثيل اللائحة المترابطة، فيمكننا بإدخال كسبيتين (هما الاجتماع بحسب المرتبة، وضغط المسار) الوصولُ إلى بنية معطيات أمثلية للمجموعات المنفصلة (بالمقاربة).

ننفِّذ العمليات الثلاث على المجموعات المنفصلة كما يلي. تُنشئ عمليةُ MAKE-SET شجرةً ذات عقدة واحدة فقط ببساطة. ننفِّذ عمليةَ FIND-SET بتتبُّع مؤشرات الأب إلى أن نجد جذر الشجرة. تؤلِّف العقدُ التي جرت زيارتها على هذا المسار البسيط إلى الجذر ***مسار الإيجاد find path***. تتسبَّب عمليةُ UNION الموضَّحة في الشكل 4.21(ب) في أن يقوم جذرُ إحدى الأشجار بالتأشير إلى جذر شجرة أخرى.



الشكل 4.21 غابة مجموعات منفصلة. (أ) شجرتان تمثلان المجموعتين في الشكل 2.21. تمثِّل الشجرةُ اليساريةُ المجموعةَ $\{b, c, e, h\}$، حيث c هو الممثِّل، وتمثِّل الشجرةُ اليمينيةُ المجموعةَ $\{d, f, g\}$، حيث f هو الممثِّل. **(ب)** نتيجة الاجتماع $\text{UNION}(e, g)$.

كسبيات لتحسين زمن التنفيذ

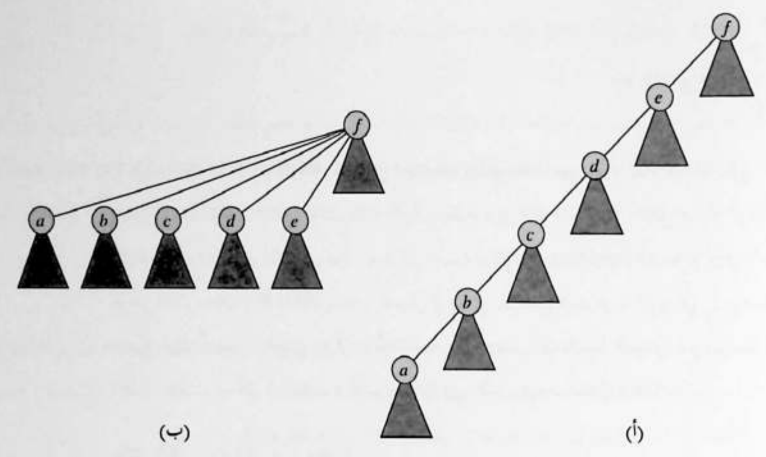
لم نحصل حتى الآن على تحسين لتنجيز اللائحة المترابطة. فمتتالية من $n - 1$ عملية UNION يمكن أن تُنشئ شجرة هي عبارة عن متتالية من n عقدة. ومع ذلك، يمكننا باستخدام كسبيتين الوصول إلى زمن تنفيذ خطي تقريبًا نسبةً إلى عدد العمليات m.

تشبه الكسبيةُ الأولى (أي ***الاجتماع بحسب المرتبة union by rank***) كسبيةَ الاجتماع المثقَّل التي استخدمناها في تمثيل اللائحة المترابطة. النهج البديهي هو أن نجعل جذر الشجرة ذات العقد الأقل يشير إلى جذر الشجرة ذات العقد الأكثر. وبدلاً من الاحتفاظ بحجم الشجرة الفرعية في كل عقدة، سنستخدم نهجًا يُسهِّل التحليل. سنحتفظ ***بمرتبةٍ rank*** لكلِّ عقدة هي حد أعلى لارتفاع العقدة. في الاجتماع بحسب المرتبة، نجعل الجذر ذا المرتبة الدنيا يشير إلى الجذر ذي المرتبة العليا خلال عملية UNION.

الكسبية الثانية (أي ***ضغط المسار path compression***) هي أيضًا غاية في البساطة وفعالة جدًّا. وكما يظهر في الشكل 5.21، فإننا نستخدمها خلال عمليات FIND-SET لجعل كل عقدة في مسار الإيجاد تشير إلى الجذر مباشرةً. لا يُغيِّر ضغط المسار أية مرتبة.

شبه رماز لغابات المجموعات المنفصلة

لتنجيز غابة من المجموعات المنفصلة باستخدام كسبية الاجتماع بحسب المرتبة، لا بدَّ أن نحتفظ بالمراتب. في كل عقدة x نحافظ على القيمة الصحيحة $x.rank$ التي هي حدٌّ أعلى لارتفاع x (عدد الوصلات في أطول مسار من ورقة منحدرة إلى x). عندما يُنشئ MAKE-SET مجموعة وحيدة العنصر تكون مرتبة هذه العقدة الوحيدة 0. لا تغيِّر عملية FIND-SET من المراتب. تتضمن عملية UNION حالتين، اعتمادًا على كون الجذرين متساويين في المرتبة أو لا. فإذا كانا غير متساويين في المرتبة، نجعل الجذر ذا المرتبة الأكبر أبًا للجذر ذي المرتبة



الشكل 5.21 ضغط المسار خلال عملية FIND-SET. جرى حذف الأسهم والحلقات الذاتية في الجذور. (أ) شجرة تمثّل مجموعة قبل تنفيذ FIND-SET(a). تمثّل المثلثات الأشجار الفرعية التي جذورها هي العقد الظاهرة. ولكل عقدة مؤشر إلى أبيها. (ب) المجموعة نفسها بعد تنفيذ FIND-SET(a). تشير كل عقدة على مسار الإيجاد الآن إلى الجذر مباشرةً.

الأصغر دون أن نغيّر المراتب. وإذا كانا متساويين في المراتب، فإننا نختار أحدهما اعتباطيًّا ليكون أبًا ونزيد مرتبته بمقدار واحد.

لنضع هذه الطريقة في شبه رماز. نرمز للأب في عقدة ما x بـ $x.p$. الإجراء LINK هو مساق فرعي يستدعيه UNION، ويأخذ مؤشرين إلى عقدتين كدخلين.

```
MAKE-SET(x)
1  x.p = x
2  x.rank = 0

UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if x.rank > y.rank
2      y.p = x
3  else x.p = y
4      if x.rank == y.rank
5          y.rank = y.rank + 1
```

الإجراء FIND-SET مع ضغط المسار هو إجراء بسيط جدًّا.

```
FIND-SET(x)
1  if x ≠ x.p
2       x.p = FIND-SET(x.p)
3  return x.p
```

الإجراء FIND-SET هو ***طريقة بمرورين two-pass method***: فهو يقوم أثناء عودته بالمرور الأول إلى أعلى مسار الإيجاد لإيجاد الجذر، وعند نشر العودية يقوم بمرور ثانٍ راجعًا إلى أسفل مسار الإيجاد لتحديث كل عقدة بحيث تشير إلى الجذر مباشرةً. يعيد كل استدعاء للإجراء FIND-SET القيمة $x.p$ في السطر 3. إذا كان x هو الجذر، فإن FIND-SET يتجاوز السطر 2 ويعيد بدلاً من ذلك $x.p$ الذي هو x؛ وهذه هي الحالة التي ينتهي فيها الصعود العودي. وإلا، يُنفَّذ السطر 2، ويعيد الاستدعاءُ العودي مع الموسط $x.p$ مؤشرًا إلى الجذر. يحدِّث السطرُ 2 العقدةَ x لتُشير مباشرةً إلى الجذر، ويعيد السطرُ 3 هذا المؤشر.

تأثير الكسبيات على زمن التنفيذ

يحسِّن الاجتماع بحسب المرتبة وضغط المسار، كلٌّ منهما على حدة، زمنَ تنفيذ العمليات على غابات المجموعات المنفصلة، ويكون هذا التحسين أعظم عند استخدامهما معًا. فالاجتماع بحسب المرتبة وحده يعطي زمن تنفيذ $O(m \lg n)$ (انظر التمرين 21.4-4)، وهذا الحد مُحكَم (انظر التمرين 21.3-3). ومع أننا لن نثبت ذلك هنا إلا أنَّه في حالة n عملية MAKE-SET (ومن ثمَّ $n-1$ عملية UNION على الأكثر) و f عملية FIND-SET، فإنَّ ضغط المسار وحده يعطي زمنًا تنفيذيًّا في أسوأ الحالات $\Theta(n + f \cdot (1 + \log_{2+f/n} n))$.

عندما نستخدم كلاً من الاجتماع وضغط المسار يكون زمن التنفيذ في أسوأ الحالات $O(m\, \alpha(n))$، حيث $\alpha(n)$ هو دالة بطيئة النمو جدًّا، نعرِّفها في المقطع 21.4. في أي تطبيق يمكن تخيُّله لبنية معطيات المجموعات المنفصلة $\alpha(n) \le 4$؛ لذا يمكننا النظر إلى زمن التنفيذ على أنه خطي نسبةً إلى m في جميع الحالات العملية. ومع ذلك، فالقول الجازم هو أنه فوق خطي. نُثْبِت هذا الحدَّ الأعلى في المقطع 21.4.

تمارين

21.3-1

أعد حل التمرين 21.2-2 باستخدام غابة مجموعات منفصلة مع الاجتماع بحسب المرتبة وضغط المسار.

21.3-2

اكتب إصدارًا غير عودي من FIND-SET مع ضغط المسار.

21.3-3

أعطِ متتالية من m عملية MAKE-SET و UNION و FIND-SET، بحيث تكون فيها n عملية MAKE-SET،

وبحيث تستغرق زمنًا $\Omega(m \lg n)$ عندما نستخدم الاجتماع بحسب المرتبة فقط.

4-3.21

افترض أننا نرغب بإضافة العملية $\text{PRINT-SET}(x)$ التي تطبع جميع عناصر مجموعة x بأي ترتيب. بيِّن كيف يمكننا إضافة واصفة وحيدة فقط لكل عقدة في غابة مجموعات منفصلة بحيث يستغرق $\text{PRINT-SET}(x)$ زمنًا خطيًّا مع عدد عناصر مجموعة x ولا تتغير أزمنة التنفيذ المقارب لبقية العمليات. افترض أننا نستطيع طباعة كل عنصر في المجموعة بزمن $O(1)$.

★ ***5-3.21***

بيِّن أنَّ أية متتالية من m عملية MAKE-SET و FIND-SET و LINK حيث تَظهر جميع عمليات LINK قبل أية عملية FIND-SET، تستغرق زمنًا $O(m)$ فقط، إذا استخدمنا كلاًّ من ضغط المسار والاجتماع بحسب المرتبة. ما الذي يجري في الحالة نفسها إذا استخدمنا كسبية ضغط المسار لوحدها؟

★ 4.21 تحليل الاجتماع بحسب المرتبة وضغط المسار

وجدنا في المقطع 3.21 أن ضمَّ الاجتماع بحسب المرتبة إلى ضغط المسار يستغرق زمن تنفيذ $O(m\,\alpha(n))$ في حالة m عملية على المجموعات المنفصلة على n عنصرًا. سنفحص في هذا المقطع الدالة α لمعرفة مدى بطء نموها. ثم نثبت زمن التنفيذ هذا باستخدام طريقة الكمون في التحليل المخمَّد.

دالة سريعة النمو جدًّا ودالتها العكسية البطيئة النمو جدًّا

نعرِّف الدالة

$$A_k(j) = \begin{cases} j+1 & \text{if } k = 0\ , \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1\ , \end{cases}$$

للأعداد الصحيحة $k \geq 0$ و $j \geq 1$، حيث يَستخدم التعبيرُ $A_{k-1}^{(j+1)}(j)$ تدوينَ التكرار الدالي المعطى في المقطع 2.3. ويكون تحديدًا $A_{k-1}^{(0)}(j) = j$ و $A_{k-1}^{(i)}(j) = A_{k-1}(A_{k-1}^{(i-1)}(j))$ لكل $i \geq 1$. سنعتبر الموسط k هو ***مستوى level*** الدالة A.

تزداد الدالة $A_k(j)$ تمامًا بازدياد j و k معًا. ولمعرفة مدى سرعة نمو هذه الدالة نحصل أولاً على تعبيرين من الشكل المغلق لكلٍّ من $A_1(j)$ و $A_2(j)$.

توطئة 2.21

إن $A_1(j) = 2j + 1$، لأي عدد صحيح $j \geq 1$.

البرهان نستخدم أولاً الاستقراء على i لبيان أنَّ $A_0^{(i)}(j) = j + i$. في الحالة الأساسية لدينا $A_0^{(0)}(j) = j = j + 0$. نفترض للخطوة الاستقرائية أنَّ $A_0^{(i-1)}(j) = j + (i-1)$. فيكون $A_0^{(i)}(j) = A_0(A_0^{(i-1)}(j)) = (j + (i-1)) + 1 = j + i$. نلاحظ أخيرًا أنَّ $A_1(j) = A_0^{(j+1)}(j) = j + (j+1) = 2j + 1$. ■

توطئة 3.21

إن $A_2(j) = 2^{j+1}(j+1) - 1$ لأي عدد صحيح $j \geq 1$.

البرهان نستخدم أولاً الاستقراء على i لبيان أنَّ $A_1^{(i)}(j) = 2^i(j+1) - 1$. في الحالة الأساسية لدينا $A_1^{(0)}(j) = j = 2^0(j+1) - 1$. نفترض للخطوة الاستقرائية أنَّ $A_1^{(i-1)}(j) = 2^{i-1}(j+1) - 1$. فيكون

$$\begin{aligned} A_1^{(i)}(j) = A_1(A_1^{(i-1)}(j)) &= A_1(2^{i-1}(j+1) - 1) = 2 \cdot \left(2^{i-1}(j+1) - 1\right) + 1 \\ &= 2^i(j+1) - 2 + 1 = 2^i(j+1) - 1 \end{aligned}$$

نلاحظ أخيرًا أنَّ $A_2(j) = A_1^{(j+1)}(j) = 2^{j+1}(j+1) - 1$. ■

يمكننا الآن ملاحظة مدى سرعة نمو $A_k(j)$ بتفحُّص $A_k(1)$ للمستويات $k = 0, 1, 2, 3, 4$ فقط. من تعريف $A_0(k)$ والتوطئتين السابقتين لدينا $A_0(1) = 1 + 1 = 2$ و $A_1(1) = 2 \cdot 1 + 1 = 3$ و $A_2(1) = 2^{1+1} \cdot (1+1) - 1 = 7$. ولدينا أيضًا:

$$\begin{aligned} A_3(1) &= A_2^{(2)}(1) \\ &= A_2(A_2(1)) \\ &= A_2(7) \\ &= 2^8 \cdot 8 - 1 \\ &= 2^{11} - 1 \\ &= 2047 \end{aligned}$$

و

$$\begin{aligned} A_4(1) &= A_3^{(2)}(1) \\ &= A_3(A_3(1)) \\ &= A_3(2047) \\ &= A_2^{(2048)}(2047) \\ &\gg A_2(2047) \\ &= 2^{2048} \cdot 2048 - 1 \\ &> 2^{2048} \end{aligned}$$

$$= (2^4)^{512}$$
$$= 16^{512}$$
$$\gg 10^{80} ,$$

وهو العدد المتوقع للذرات في الكون المنظور. (يمثل الرمز ≫ علاقة "أكبر بكثير من".)

نعرّف الدالة المعاكسة للدالة $A_k(n)$ لكل عدد صحيح $n \geq 0$ كما يلي:

$$\alpha(n) = \min\{k : A_k(1) \geq n\}$$

باختصار $\alpha(n)$ هو المستوى الأدنى k الذي يكون من أجله $A_k(1)$ هو n على الأقل. نرى من قيم $A_k(1)$ المذكورة آنفًا أنَّ

$$\alpha(n) = \begin{cases} 0 & \text{for } 0 \leq n \leq 2 \ , \\ 1 & \text{for } n = 3 \ , \\ 2 & \text{for } 4 \leq n \leq 7 \ , \\ 3 & \text{for } 8 \leq n \leq 2047 \ , \\ 4 & \text{for } 2048 \leq n \leq A_4(1) \ . \end{cases}$$

أي تكون $\alpha(n) > 4$ فقط لقيم كبيرة جدًّا لدرجةٍ "فلكية" لـ n (أكبر من $A_4(1)$، رقم هائل جدًّا)، وبذلك تكون $\alpha(n) \leq 4$ لجميع الأغراض العملية.

خصائص المراتب

نُثْبِت فيما تبقى من هذا المقطع أنَّ $O(m\,\alpha(n))$ هي حدٌّ لزمن التنفيذ في العمليات على المجموعات المنفصلة باستخدام الاجتماع بحسب المرتبة وضغط المسار. بهدف إثبات هذا الحد، نبدأ أولاً بإثبات بعض خصائص المراتب.

توطئة 4.21

إن $x.rank \leq x.p.rank$ مع متراجحة تامة إذا كان $x \neq x.p$، وذلك لجميع العقد x. تكون قيمة $x.rank$ في البداية 0 وتزداد مع الوقت إلى أن يصبح $x \neq x.p$؛ ولا تتغير بعد ذلك قيمة $x.rank$. تزداد قيمة $x.p.rank$ باطراد مع مرور الزمن.

البرهان تُبرهَن هذه التوطئة بالاستقراء المباشر على عدد العمليات باستخدام تنجيزات MAKE-SET و UNION و FIND-SET التي تظهر في المقطع 3.21. سنترك البرهان للتمرين 4.21-1. ■

نتيجة 5.21

عندما نتبع المسار البسيط من أية عقدة إلى الجذر، فإنَّ مراتب العقد تتزايد تمامًا. ■

توطئة 6.21

مرتبة كل عقدة هي $n-1$ على الأكثر.

البرهان تبدأ مرتبة كل عقدة من 0 وتزداد في عمليات LINK فقط. وبسبب وجود $n-1$ عملية UNION على الأكثر، توجد $n-1$ عملية LINK على الأكثر. ولمّا كانت كلُّ عملية LINK إما أن تُبقِيَ المراتب على ما هي عليه وإما أن تزيد مراتب بعض العقد بمقدار 1، فإنَّ كلَّ المراتب هي $n-1$ على الأكثر. ■

توفِّر التوطئة 6.21 حدًّا ضعيفًا على المراتب. والحقيقة أنَّ مرتبة كل عقدة هي $\lfloor \lg n \rfloor$ على الأكثر (انظر التمرين 21.4-2). ومع ذلك، فإنَّ الحد المنحل للتوطئة 6.21 سيكفي لتحقيق أهدافنا.

إثبات الحد الزمني

سنستخدم طريقة الكمون في التحليل المخمَّد (انظر المقطع 3.17) لإثبات الحد الزمني $O(m\,\alpha(n))$. عند إجراء تحليل مخمَّد سنجد أن من المناسب أن نفترض أننا نستدعي عملية LINK بدلاً من عملية UNION. وذلك لأنَّ موسطي الإجراء LINK هما مؤشران إلى جذرين، بافتراض أنَّنا نُنْجِز عمليات FIND-SET بصورة مستقلة. تُبيِّن التوطئة التالية أننا حتى لو أخذنا بالحسبان عمليات FIND-SET الإضافية التي تنتج عن استدعاءات UNION، فإنَّ زمن التنفيذ المقارب لا يتغيَّر.

توطئة 7.21

افترض أننا نحوِّل متتالية S' من m' عملية MAKE-SET و UNION و FIND-SET إلى متتالية S من m عملية MAKE-SET و LINK و FIND-SET بتحويل كلّ UNION إلى عمليتي FIND-SET تتلوهما عملية LINK. عندئذٍ، إذا كانت المتتالية S تُنفَّذ بزمن $O(m\,\alpha(n))$، فإنَّ المتتالية S' تُنفَّذ بزمن $O(m'\,\alpha(n))$.

البرهان لمّا كانت كل عملية UNION في المتتالية S' تُحوَّل إلى ثلاث عمليات في S، فلدينا $m' \le m \le 3m'$. ولما كان $m = O(m')$، فإنَّ الحد الزمني للمتتالية المحوَّلة $O(m\,\alpha(n))$ يقتضي حدًّا زمنيًّا $O(m'\,\alpha(n))$ للمتتالية الأصلية S'. ■

سنفترض فيما تبقى من هذا المقطع أنَّ المتتالية البدائية المؤلفة من m' عملية MAKE-SET و UNION و FIND-SET قد جرى تحويلها إلى متتالية من m عملية MAKE-SET و LINK و FIND-SET. ونثبت الآن حدًّا زمنيًّا $O(m\,\alpha(n))$ للمتتالية المحوَّلة، ونعتمد على التوطئة 7.21 لإثبات زمن تنفيذ $O(m'\,\alpha(n))$ للمتتالية الأصلية ذات الـ m' عملية.

دالة الكمون

تسند دالة الكمون التي نستخدمها كمونًا $\phi_q(x)$ لكل عقدة x في غابة المجموعات المنفصلة بعد q عملية. نجمع كمون العقد لنحصل على كمون الغابة كاملةً: $\Phi_q = \sum_x \phi_q(x)$، حيث يرمز Φ_q إلى كمون الغابة بعد q عملية. تكون الغابة فارغةً قبل العملية الأولى، ونختار $\Phi_0 = 0$. ولن يكون أيُّ كمون Φ_q سالبًا أبدًا.

تعتمد قيمة $\phi_q(x)$ على كون x جذرًا للشجرة بعد العملية q. فإذا كان كذلك، أو كان $x.rank = 0$، فإن $\phi_q(x) = \alpha(n) \cdot x.rank$.

افترض الآن أنه بعد العملية q لن يكون x جذرًا، وأن $x.rank \geq 1$. نحتاج إلى تعريف دالتين مساعدتين على x قبل أن نستطيع تعريف $\phi_q(x)$. نعرِّف أولاً

$$\text{level}(x) = \max \{k : x.p.rank \geq A_k(x.rank)\} .$$

أي إنَّ $\text{level}(x)$ هو المستوى الأعظم k الذي لا يكون من أجله A_k المطبق على مرتبة x أكبر من مرتبة أبي x.

إنَّ

$$0 \leq \text{level}(x) < \alpha(n) , \tag{1.21}$$

التي ننظر إليها كما يلي. لدينا

$$\begin{aligned} x.p.rank &\geq x.rank + 1 && \text{(بحسب التوطئة (4.21))} \\ &= A_0(x.rank) , && \text{(بحسب تعريف } A_0(j)\text{)} \end{aligned}$$

وهذا يقتضي أنَّ $\text{level}(x) \geq 0$ ويكون

$$\begin{aligned} A_{\alpha(n)}(x.rank) &\geq A_{\alpha(n)}(1) && \text{(لأن } A_k(j) \text{ متزايد تماما)} \\ &\geq n && \text{(بحسب تعريف } \alpha(n)\text{)} \\ &> x.p.rank , && \text{(بحسب التوطئة (6.21))} \end{aligned}$$

وهذا بدوره يقتضي أن يكون $\text{level}(x) < \alpha(n)$. لاحظ أنه لـمّا كان $x.p.rank$ متزايد باطراد مع الزمن، فإنَّ $\text{level}(x)$ هو متزايد باطراد أيضًا.

تُطبَّق الدالة المساعدة الثانية عندما يكون $x.rank \geq 1$

$$\text{iter}(x) = \max \{i : x.p.rank \geq A_{\text{level}(x)}^{(i)}(x.rank)\} .$$

أي إنَّ $\text{iter}(x)$ هو أكبر عدد من المرات التي يمكننا فيها تطبيق $A_{\text{level}(x)}$ المطبَّق بدايةً على مرتبة x، قبل أن نحصل على قيمة أكبر من مرتبة أبي x.

عندما تكون $x.rank \geq 1$ يكون لدينا:

$$1 \leq \text{iter}(x) \leq x.rank\ , \qquad (2.21)$$

التي ننظر إليها كما يلي. لدينا

$$x.p.rank \ \geq \ A_{\text{level}(x)}(x.rank) \qquad \text{(بحسب تعريف } \text{level}(x)\text{)}$$
$$= A^{(1)}_{\text{level}(x)}(x.rank) \qquad \text{(بحسب تعريف التكرار الوظيفي)}$$

وهذا يقتضي أنَّ $\text{iter}(x) \geq 1$ ويكون لدينا

$$A^{(x.rank+1)}_{\text{level}(x)}(x.rank) \ = \ A_{\text{level}(x)+1}(x.rank) \qquad \text{(بحسب تعريف } A_k(j)\text{)}$$
$$> x.p.rank\ , \qquad \text{(بحسب تعريف } \text{level}(x)\text{)}$$

وهذا بدوره يقتضي أنَّ $\text{iter}(x) \leq x.rank$. لاحظ أنه لـمّا كان $x.p.rank$ متزايد باطراد مع الزمن (لكي يتناقص $\text{iter}(x)$)، فإن $\text{level}(x)$ يجب أن يتزايد. ومع بقاء $\text{level}(x)$ دون تغيير، فإنَّ $\text{iter}(x)$ إما أن يتزايد وإما أن يبقى دون تغيير.

بعد تعريف هاتين الدالتين، نعرِّف كمون عقدة x بعد q عملية:

$$\phi_q(x) = \begin{cases} \alpha(n) \cdot x.rank & \text{if } x \text{ is a root or } x.rank = 0\ , \\ (\alpha(n) - \text{level}(x)) \cdot x.rank - \text{iter}(x) & \text{if } x \text{ is not a root and } x.rank \geq 1\ . \end{cases}$$

تعطي التوطئتان التاليتان خواصّ مفيدة لكمونات العقد.

توطئة 8.21

$$0 \leq \phi_q(x) \leq \alpha(n) \cdot x.rank$$

لكل عقدة x ولكل أعداد العمليات q.

البرهان إذا كان x جذرًا أو كان $x.rank = 0$، فإن $\phi_q(x) = \alpha(n).x.rank$ بالتعريف. افترض الآن أنَّ x ليس جذرًا وأنَّ $x.rank \geq 1$. نحصل على حد أدنى لـ $\phi_q(x)$ بتعظيم قيم $\text{level}(x)$ و $\text{iter}(x)$. وبحسب الحد (1.21) يكون $\text{level}(x) \leq \alpha(n) - 1$، وبحسب الحد (2.21) يكون $\text{iter}(x) \leq x.rank$. وبذلك يكون

$$\begin{aligned} \phi_q(x) &= (\alpha(n) - \text{level}(x)) \cdot x.rank - \text{iter}(x) \\ &\geq (\alpha(n) - (\alpha(n) - 1) \cdot x.rank - x.rank \\ &= x.rank - x.rank \\ &= 0\ . \end{aligned}$$

وبالمثل نحصل على حد أعلى على $\phi_q(x)$ بتصغير قيم $\text{level}(x)$ و $\text{iter}(x)$. وبحسب الحد (1.21) يكون

$\text{level}(x) \geq 0$، وبحسب الحد (2.21) يكون $\text{iter}(x) \geq 1$. وبذلك يكون

$$\begin{aligned}\phi_q(x) &\leq (\alpha(n) - 0) \cdot x.rank - 1 \\ &= \alpha(n) \cdot x.rank - 1 \\ &< \alpha(n) \cdot x.rank\ .\end{aligned}$$

■

نتيجة 9.21

إذا لم تكن العقدة جذرًا، وكان $x.rank > 0$، فإن $\phi_q(x) < \alpha(n) \cdot x.rank$. ■

تغيُّرات الكمون والتكلفة المخمَّدة للعمليات

لندرس الآن تأثير عمليات المجموعات المنفصلة على كمونات العقد. إن معرفة التغيير في الكمون نتيجة كل عملية، يمكّننا من تحديد التكلفة المخمَّدة لها.

توطئة 10.21

لتكن x عقدة ليست جذرًا، ولنفترض أنَّ العملية q هي LINK أو FIND-SET. فيكون لدينا بعد العملية ذات الرقم q $\phi_q(x) \leq \phi_{q-1}(x)$. وإذا كان $x.rank \geq 1$ ، وتغيَّر $\text{level}(x)$ أو $\text{iter}(x)$ بسبب العملية q، فإن $\phi_q(x) \leq \phi_{q-1}(x) - 1$. أي إنَّ كمون x لا يمكن أن يزداد، وإذا كان له مرتبة موجبة وتغيّر $\text{level}(x)$ أو $\text{iter}(x)$، فإنَّ كمون x ينقص بمقدار 1 على الأقل.

البرهان إن x ليست جذرًا، لذا فإنَّ العملية ذات الرقم q لا تغيِّر $x.rank$، ولمّا كانت n لا تتغيَّر بعد عملية MAKE-SET بدائية، فإنَّ $\alpha(n)$ يبقى أيضًا دون تغيير، لأن هذه المركبات في صيغة كمون x تبقى نفسها بعد العملية q. فإذا كان $x.rank = 0$، فإن $\phi_q(x) = \phi_{q-1}(x) = 0$. افترض الآن أنَّ $x.rank \geq 1$.

تذكَّر أنَّ $\text{level}(x)$ تتزايد باطراد مع الزمن. فإذا تركت العمليةُ ذات الرقم q القيمةَ $\text{level}(x)$ دون تغيير، ازداد $\text{iter}(x)$ أو بقي دون تغيير. وإذا لم يتغير أيٌّ من $\text{level}(x)$ و $\text{iter}(x)$، فإن $\phi_q(x) = \phi_{q-1}(x)$. وإذا لم يتغير $\text{level}(x)$ وازداد $\text{iter}(x)$، فإنه يزداد بمقدار 1 على الأقل، وبذلك يكون $\phi_q(x) \leq \phi_{q-1}(x) - 1$.

أخيرًا، إذا زادت العملية ذات الرقم q من $\text{level}(x)$، فإنه يزداد بمقدار 1 على الأقل، أي إنَّ قيمة الحد $(\alpha(n) - \text{level}(x)) \cdot x.rank$ تنخفض بمقدار $x.rank$ على الأقل. ولمّا كان $\text{level}(x)$ قد ازداد، فإنَّ قيمة $\text{iter}(x)$ قد تنخفض، لكن بحسب الحد (2.21)، يكون الانخفاض بمقدار $x.rank - 1$. أي إنَّ زيادة الكمون نتيجةً لتغيير $\text{iter}(x)$ هي أقل من نقصان الكمون نتيجة لتغيير $\text{level}(x)$، ونستنتج أنَّ $\phi_q(x) \leq \phi_{q-1}(x) - 1$. ■

تبيِّن التوطئات الثلاث الأخيرة أنَّ الكمون المخمَّد لكل عملية من العمليات MAKE-SET و LINK و FIND-SET هو $O(\alpha(n))$. تذكَّر من المعادلة (2.17) أنَّ التكلفة المخمَّدة لكل عملية هي تكلفتها الفعلية يُضاف إليها الزيادة في الكمون نتيجةً للعملية.

توطئة 11.21

التكلفة المخمَّدة لكل عملية MAKE-SET هي $O(1)$.

البرهان افترض أنَّ العملية q هي MAKE-SET(x). تنشئ هذه العملية عقدة x مرتبتها 0، أي $\phi_q(x) = 0$، دون أن تتغير بقية المراتب أو الكمونات، وبذلك يكون $\Phi_q = \Phi_{q-1}$. وبملاحظة أنَّ التكلفة الفعلية لعملية MAKE-SET هي $O(1)$ يكتمل البرهان. ■

توطئة 12.21

التكلفة المخمَّدة لكل عملية LINK هي $O(\alpha(n))$.

البرهان افترض أنَّ العملية q هي LINK(x,y). التكلفة الفعلية لعملية LINK هي $O(1)$. دون فقد للعمومية، افترض أنَّ عملية LINK تجعل y أبًا لـ x.

لتحديد التغيير في الكمون بنتيجة عملية LINK، نلاحظ أنَّ العقد الوحيدة التي يمكن أن تتغير هي x و y وأبناء y قبل العملية مباشرةً. سنبيّن أنَّ العقدة الوحيدة التي يمكن أن يزيد كمونها بنتيجة عملية LINK هي y، وأنَّ هذه الزيادة هي على الأكثر $\alpha(n)$:

- بحسب التوطئة 10.21 فإنَّ أي عقدة كانت ابنًا لـ y قبل عملية LINK مباشرةً لا يمكن أن يزداد كمونها بنتيجة العملية LINK.

- من تعريف $\phi_q(x)$ نرى أنَّه لمّا كان x جذرًا قبل العملية q مباشرةً فإنَّ $\phi_{q-1}(x) = \alpha(n) \cdot x.rank$ فإذا كان $x.rank = 0$، فإن $\phi_q(x) = \phi_{q-1}(x) = 0$. وإلاَّ فإن

$$\begin{aligned}\phi_q(x) &< \alpha(n) \cdot x.rank \qquad (\text{بحسب النتيجة } (9.21))\\ &= \phi_{q-1}(x)\,,\end{aligned}$$

ومن ثمَّ فإنَّ كمون x ينقص.

- لـمّا كان y جذرًا قبل عملية LINK، فإنَّ $\phi_{q-1}(y) = \alpha(n) \cdot y.rank$. وتُبقي عمليةُ LINK y جذرًا، ثم إنها إما أن تُبقي مرتبة y على ما هي عليه، وإما أن تزيدها بمقدار 1. لذا فإن $\phi_q(y) = \phi_{q-1}(y)$ أو $\phi_q(y) = \phi_{q-1}(y) + \alpha(n)$.

فالزيادة في الكمون بنتيجة العملية LINK هي إذن $\alpha(n)$ على الأكثر. والتكلفة المخمَّدة لعملية LINK هي $O(1) + \alpha(n) = O(\alpha(n))$. ■

توطئة 13.21

التكلفة المخمَّدة لكل عملية FIND-SET هي $O(\alpha(n))$.

البرهان افترض أنَّ العملية q هي FIND-SET وأنَّ مسار الإيجاد يحتوي s عقدة. إن التكلفة الفعلية لعملية FIND-SET هي $O(s)$. سنبيّن أن كمون العقد لا يزداد بنتيجة عملية FIND-SET وأنَّ هناك $\max(0,\ s-(\alpha(n)+2))$ عقدة على الأقل على مسار الإيجاد ينقص كمونها بمقدار 1 على الأقل.

لبيان أن كمون العقد لا يزداد، نلجأ أولاً إلى التوطئة 10.21 لجميع العقد ما عدا الجذر. إذا كان x هو الجذر، فإنَّ كمونه هو $\alpha(n)\cdot x.rank$، وهو لا يتغير.

نبيِّن الآن أنَّ هناك $\max(0,\ s-(\alpha(n)+2))$ عقدة على الأقل ينقص كمونها بمقدار 1 على الأقل. لتكن x عقدة على مسار الإيجاد بحيث $x.rank > 0$ ويتبعها في مكان ما من مسار الإيجاد عقدة أخرى y ليست جذرًا، بحيث يكون $\text{level}(y) = \text{level}(x)$ قبل عملية FIND-SET مباشرةً. (لاحظ أنه ليس بالضرورة أن تكون العقدة y تتبع العقدة x مباشرةً على مسار الإيجاد.) جميع العقد على مسار الإيجاد عدا $\alpha(n)+2$ عقدة تحقق هذه القيود على x. فالعقد التي لا تحققها هي العقدة الأولى على مسار الإيجاد (إذا كانت مرتبتها 0)، والعقدة الأخيرة على المسار (أي الجذر)، والعقدة الأخيرة w على المسار التي يكون عندها $\text{level}(w) = k$، في حال $k = 0, 1, 2, \ldots, \alpha(n)-1$.

نثبِّت هذه العقدة x، ونبيّن أنَّ كمونها ينقص بمقدار 1 على الأقل. ليكن $k = \text{level}(x) = \text{level}(y)$. يكون لدينا قبل ضغط المسار الذي تسببه FIND-SET مباشرةً

$$x.p.rank \geq A_k^{\text{iter}(x)}(x.rank) \qquad \text{(بحسب تعريف iter}(x))$$

$$y.p.rank \geq A_k(y.rank) \qquad \text{(بحسب تعريف iter}(y))$$

$$y.rank \geq x.p.rank\ . \qquad \text{(بحسب النتيجة (5.21) ولأنَّ } y \text{ تتبع } x \text{ على مسار الإيجاد)}$$

بوضع هذه المتراجحات معًا، وبافتراض أن i هي قيمة $\text{iter}(x)$ قبل ضغط المسار، يكون لدينا

$$\begin{aligned} y.p.rank &\geq A_k(y.rank) \\ &\geq A_k(x.p.rank) && \text{(لأن } A_k(j) \text{ متزايد تماما)} \\ &\geq A_k\left(A_k^{\text{iter}(x)}(x.rank)\right) \\ &= A_k^{i+1}(x.rank)\ . \end{aligned}$$

لمّا كان ضغط المسار سيجعل لكلٍّ من x و y الأب نفسه، فإننا نعلم أنه بعد ضغط المسار سيكون $x.p.rank = y.p.rank$ وأنَّ ضغط المسار لا يُنقِص $y.p.rank$. ولمّا كان $x.rank$ لا يتغير بعد ضغط المسار فلدينا $x.p.rank \geq A_k^{i+1}(x.rank)$. وهكذا، فإنَّ ضغط المسار سيسبب ازدياد $\text{iter}(x)$ (إلى $i+1$ على الأقل) أو ازدياد $\text{level}(x)$ (الذي يحدث إذا ازداد $\text{iter}(x)$ إلى القيمة $x.rank + 1$ على الأقل). في كلا الحالتين، وبحسب التوطئة 10.21، يكون لدينا $\phi_q(x) \leq \phi_{q-1}(x) - 1$. ومن ثمَّ فإنَّ كمون x ينقص بمقدار 1 على الأقل.

إن التكلفة المخمَّدة لعملية FIND-SET هي التكلفة الفعلية إضافةً إلى التغيُّر في الكمون. والتكلفة الفعلية هي $O(s)$، وقد بيَّنا أنَّ الكمون الكلي ينقص بمقدار $\max(0, s - (\alpha(n) + 2))$. وبذلك تكون التكلفة المخمَّدة هي على الأكثر $O(s) - (s - (\alpha(n) + 2)) = O(s) - s + O(\alpha(n)) = O(\alpha(n))$، لأننا نستطيع رفع وحدات الكمون لتطغى على الثابت المضمر في $O(s)$. ■

ينتج عن التوطئات السابقة جميعها المبرهنة التالية.

مبرهنة 14.21

يمكن تنفيذ متتالية من m عملية MAKE-SET و UNION و FIND-SET من بينها n عملية MAKE-SET على غابة من المجموعات المنفصلة باستخدام الاجتماع المثقَّل وضغط المسار في أسوأ الحالات بزمن $O(m\,\alpha(n))$.

البرهان يتحقَّق البرهان مباشرة من التوطئات 7.21 و 11.21 و 12.21 و 13.21. ■

تمارين

1-4.21

أثبت التوطئة 4.21.

2-4.21

أثبت أنَّ مرتبة أية عقدة هي $\lfloor \lg n \rfloor$ على الأكثر.

3-4.21

على ضوء التمرين 2-4.21، كم عدد البتات الضرورية لخزن $x.rank$ لكلِّ عقدة x؟

4-4.21

باستخدام التمرين 2-4.21، أعطِ برهانًا بسيطًا على أنَّ العمليات على غابة المجموعات المنفصلة مع الاجتماع بحسب المرتبة ولكن بدون ضغط المسار تُنفَّذ بزمن $O(m \lg n)$.

5-4.21

يعتقد الأستاذ Dante أنَّه لمّا كانت مراتب العقد تتزايد تمامًا على مسار بسيط إلى الجذر، فإنَّ مستويات العقد يجب أن تتزايد باطراد على ذلك المسار. بتعبير آخر، إذا كان $x.rank > 0$ ولم يكن $x.p$ جذرًا، فإنَّ $\text{level}(x) \le \text{level}(x.p)$. هل الأستاذ على صواب؟

★ ***6-4.21***

لتكن لدينا الدالة $\alpha'(n) = \min\{k : A_k(1) \ge \lg(n+1)\}$. بيِّن أنَّ $\alpha'(n) \le 3$ لجميع القيم العملية لـ n، وبيّن باستخدام التمرين 2-4.21 كيف يمكن تعديل محدد دالة الكمون لإثبات أنَّه يمكننا تنفيذ متتالية من m عملية MAKE-SET و UNION و FIND-SET من بينها n عملية MAKE-SET على غابة من المجموعات المنفصلة باستخدام الاجتماع بحسب المرتبة وضغط المسار في أسوأ الحالات بزمن $O(m\,\alpha'(n))$.

مسائل

1-21 الحد الأصغر خارج الخط

المطلوب في ***مسألة الحد الأصغر خارج الخط*** ***off-line minimum problem*** أن نحافظ على مجموعة ديناميكية T من العناصر من المجال $\{1,2,\ldots,n\}$ مع العمليات INSERT و EXTRACT-MIN . لدينا متتالية S من n استدعاء INSERT و m استدعاء EXTRACT-MIN، حيث يجري إدراج كل مفتاح من $\{1,2,\ldots,n\}$ مرةً واحدة. نرغب بتحديد المفتاح الذي يعيده كل استدعاء EXTRACT-MIN. ونرغب تحديدًا بملء صفيفة $extracted[1\,..\,m]$، حيث $extracted[i]$ هو المفتاح الذي يعيده الاستدعاء i لـ EXTRACT-MIN لكل $i = 1,2,\ldots,m$. تعني مسألة "خارج الخط" أنه يمكننا معالجة المتتالية S كاملةً قبل تحديد أي من المفاتيح المعادة.

أ. في المتنسخ (instance) التالي من مسالة الحد الأصغر خارج الخط، تمثّل كل عملية INSERT(i) بقيمة i وتمثّل كل EXTRACT-MIN بحرف E:

4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5 .

املأ القيم الصحيحة في الصفيفة *extracted*.

لإنشاء خوارزمية لهذه المسألة، نقسّم المتتالية S إلى متتاليات جزئية متجانسة. أي نمثّل S كما يلي:

$I_1, E, I_2, E, I_3, \ldots, I_m, E, I_{m+1}$,

حيث تمثّل كل E استدعاءً واحدًا لـ EXTRACT-MIN ويمثّل كل I_j متتالية من استدعاءات INSERT (قد تكون فارغة). نضع في البداية لكلِّ متتاليةٍ جزئيةٍ I_j المفاتيحَ المدرجةَ في هذه العمليات ضمن مجموعة K_j (تكون فارغة إذا كانت I_j فارغة). ثم نقوم بما يلي:

```
OFF-LINE-MINIMUM(m, n)
1  for i = 1 to n
2      determine j such that i ∈ K_j
3      if j ≠ m + 1
4          extracted[j] = i
5          let l be the smallest value greater than j
                   for which set K_l exists
6          K_l = K_j ∪ K_l , destroying K_j
7  return extracted
```

ب. ناقش صحة كون الصفيفة $extracted$ التي يعيدها OFF-LINE-MINIMUM صحيحة.

ت. صِفْ كيفية تنجيز OFF-LINE-MINIMUM تنجيزًا فعَّالاً مع بنية معطيات للمجموعات المنفصلة. أعطِ حدًّا مُحْكَمًا لزمن تنفيذ تنجيزك في أسوأ الحالات.

2-21 تحديد العمق

في ***مسألة تحديد العمق depth-determination problem*** نحافظ على غابة $\mathcal{F} = \{T_i\}$ من الأشجار ذوات الجذور مع ثلاث عمليات:

MAKE-TREE(v) إنشاء شجرة ذات عقدة واحدة v.

FIND-DEPTH(v) إعادة عمق عقدة v ضمن شجرتها.

GRAFT(r, v) تجعل العقدةَ r (التي يفترض أن تكون هي جذر الشجرة) ابنًا لعقدةٍ v (التي يفترض أن تكون في شجرة أخرى غير r ولكن يمكن أن تكون جذرًا أو لا).

أ. افترض أننا نستخدم تمثيلاً للشجرة يشبه غابة المجموعات المنفصلة: $v.p$ هو أب للعقدة v، إلا إذا كان v جذرًا فيكون $v.p = v$. افترض أيضًا أنَّنا نجَّزنا GRAFT(r, v) بوضع $r.p = v$ و FIND-DEPTH(v) باتباع مسار الإيجاد إلى الجذر، وبإعادة عدد العقد غير v التي جرت مصادفتها، بيِّن أنَّ زمن التنفيذ في أسوأ الحالات لمتتالية من m عملية MAKE-TREE و FIND-SET و GRAFT هو $\Theta(m^2)$.

باستخدام كسبيتي الاجتماع بحسب المرتبة وضغط المسار يمكننا تقليص زمن التنفيذ في أسوأ الحالات. نستخدم غابة المجموعات المنفصلة $\mathcal{S} = \{S_i\}$، حيث تقابل كلُّ مجموعةٍ S_i (التي هي شجرة بحد ذاتها) شجرةً T_i في الغابة $\mathcal{F}$. ومع ذلك، فإن بنية الشجرة الخاصة بالمجموعة S_i لا تقابل بالضرورة بنية المعطيات الخاصة بـ T_i. في الحقيقة لا يُسجِّل تنجيز S_i العلاقة الدقيقة بين الأب والابن، ومع ذلك فهو يسمح بتحديد عمق أية عقدة في T_i.

الفكرة الرئيسية هي الحفاظ على "شبه مسافة" $v.d$ في كل عقدة v، التي تعرَّف بحيث يكون مجموع أشباه المسافات على المسار البسيط من v إلى جذر مجموعتها S_i مساويًا لعمق v في T_i. أي إذا كان المسار البسيط من v إلى جذرها في S_i هو $v_0, v_1, \ldots, v_k$ حيث $v_0 = v$ و v_k هو جذر S_i، فيكون عمق v في T_i هو $\sum_{j=0}^{k} v_j.d$.

ب. أعطِ تنجيزًا لـ MAKE-TREE.

ت. بيِّن كيف نعدِّل FIND-SET لتنجيز FIND-DEPTH. يجب أن يَضغط تنجيزُك المسار وأن يكون زمن تنفيذه خطيًّا بدلالة طول مسار الإيجاد. تأكَّد أنَّ تنجيزك يُحدِّث أشباهَ المسافات تحديثًا صحيحًا.

ث. بيّن كيفية تنجيز $\text{GRAFT}(r, v)$ الذي يجمع المجموعتين المتضمنتين لـ r و v بتعديل الإجراءين UNION و LINK . تأكَّد أنَّ تنجيزك يُحدِّث أشباهَ المسافات تحديثًا صحيحًا. لاحظ أنَّ جذر مجموعة S_i ليس بالضرورة جذرًا للشجرة المقابلة T_i.

ج. أعطِ حدًّا مُحْكَمًا لزمن التنفيذ في أسوأ الحالات لمتتالية من m عملية MAKE-TREE و FIND-DEPTH و GRAFT من بينها n عملية MAKE-TREE.

3-21 ***خوارزمية تاريان خارج الخط للبحث عن السلف المشترك الأبعد***

السلف المشترك الأبعد ***least common ancestor*** لعقدتين u و v في شجرة ذات جذر T هو العقدة w التي تكون سلفًا لكلٍّ من u و v ويكون لها العمق الأكبر في T. في ***مسألة الأسلاف المشتركة الأبعد خارج الخط*** ***off-line least common-ancestors problem***، لدينا شجرة T ومجموعة $P = \{\{u, v\}\}$ من الأزواج غير المرتبة من العقد في T، ونرغب بتحديد السلف المشترك الأبعد لكل زوج في P.

لحل مسألة الأسلاف المشتركة الأبعد خارج الخط، يقوم الإجراء التالي بتجوال في الشجرة T مع استدعاء ابتدائي $\text{LCA}(T.root)$. نفترض تلوين كل عقدة بالأبيض WHITE قبل التجوال.

```
LCA(u)
 1  MAKE-SET(u)
 2  FIND-SET(u).ancestor = u
 3  for each child v of u in T
 4       LCA(v)
 5       UNION(u, v)
 6       FIND-SET(u).ancestor = u
 7  u.color = BLACK
 8  for each node v such that {u, v} ∈ P
 9       if v.color == BLACK
10            print "The least common ancestor of"
                   u "and" v "is" FIND-SET(v).ancestor
```

أ. أثبت أن السطر 10 يُنفَّذ مرةً واحدة لكل زوج $\{u, v\} \in P$.

ب. بيِّن أنه عند استدعاء $\text{LCA}(u)$ يكون عدد المجموعات في بنية معطيات المجموعات المنفصلة مساويًا لعمق u في T.

ت. أثبت أنَّ LCA يطبع السلف المشترك الأبعد لـ u و v لكل زوج $\{u, v\} \in P$.

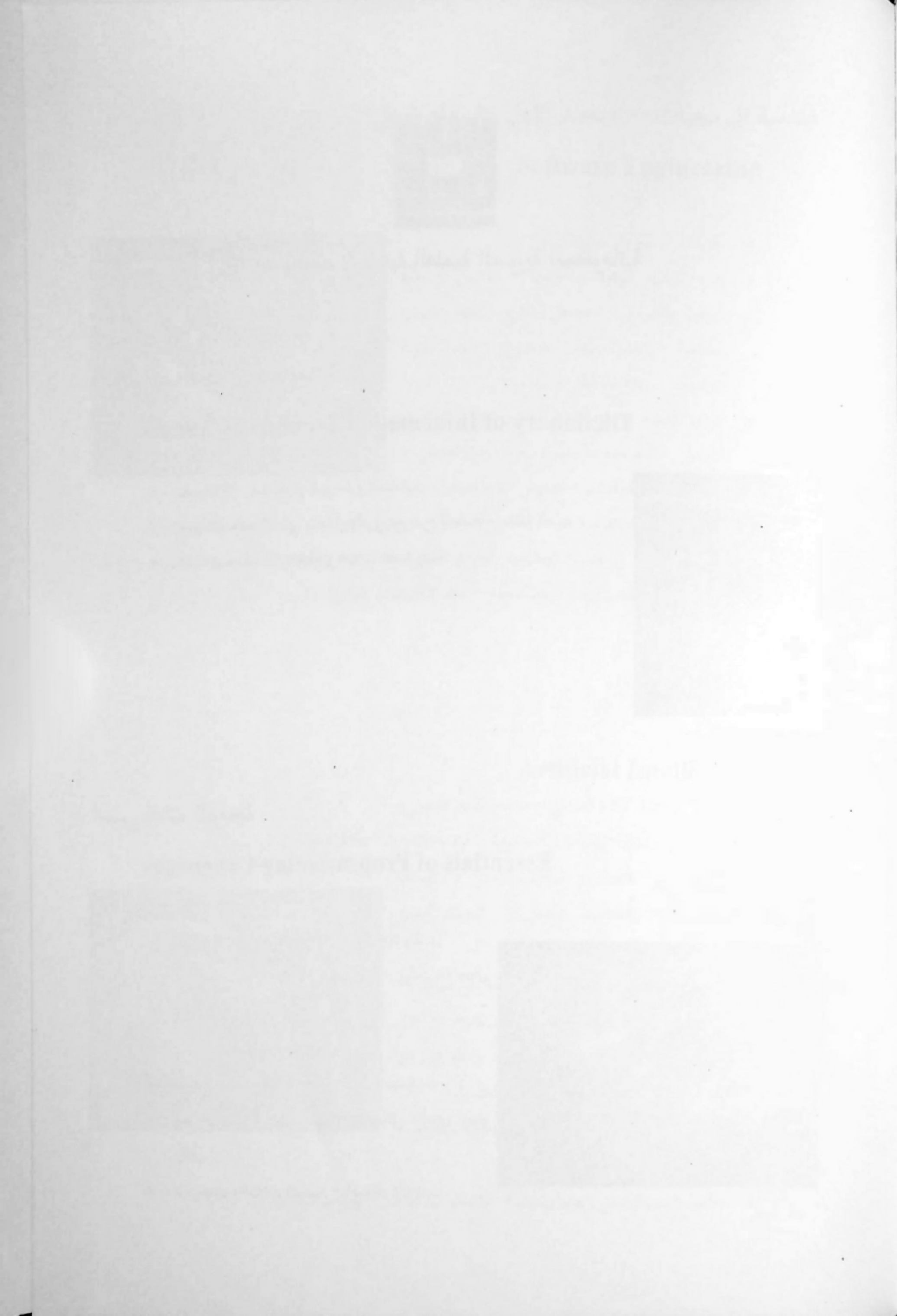
ث. حلِّل زمن تنفيذ LCA بافتراض أننا نستخدم تنجيز بنية معطيات المجموعات المنفصلة من المقطع 3.21.

ملاحظات الفصل

يعود الفضل في العديد من النتائج الهامة المتعلقة ببنية معطيات المجموعات المنفصلة (جزئيًا على الأقل) إلى تاريان R. E. Tarjan؛ فقد أعطى Tarjan [328, 330]، باستخدام التحليل التجميعي aggregate analysis، الحدَّ الأعلى المُحْكَم الأول بدلالة الدالة المعاكسة لدالة Ackermann ذات النمو البطيء جدًّا $\hat{\alpha}(m, n)$. (تشبة الدالةُ $A_k(j)$ المُعطاةُ في المقطع 4.21 دالةَ Ackermann، وتشبه الدالةُ $\alpha(n)$ الدالةَ المعاكسة. وتكون قيمة كلٍّ من $\alpha(n)$ و $\hat{\alpha}(m, n)$: 4 على الأكثر لجميع القيم المعقولة لـ m و n.) برهَنَ Hopcroft و Ullman [5, 179] على حدٍّ أعلى $O(m \lg^* n)$. جرت ملاءمة المعالجة في المقطع 4.21 بتحليل لاحق في Tarjan [332] الذي يعتمد بدوره على Kozen [220]. يعطي Harfst و Reingold [161] إصدارًا للحد السابق لـ Tarjan يعتمد على الكمون.

يناقش Tarjan و van Leeuwen [333] متغيرات لكسبية ضغط المسار، تتضمن "طرائق المرور الواحد" التي تقدّم أحيانًا عوامل ثابتة، أفضل في أدائها من الطرائق ذات المرورين. وكما في التحاليل السابقة لـ Tarjan لكسبية ضغط المسار الأساسية، فإنَّ تحاليل Tarjan و van Leeuwen هي تجميعية. بيَّن Harfst و Reingold [161] فيما بعد كيف يمكن بتغيير بسيط لدالة الكمون ملاءمة تحليل ضغط المسار للمتغيرات ذات المرور الواحد. يبيِّن Gabow و Tarjan [121] أنَّه في بعض التطبيقات، يمكن إجراء العمليات على المجموعات المنفصلة بزمن $O(m)$.

بيَّنَ Tarjan [329] ضرورة وجود حدٍّ أدنى للزمن $\Omega(m\,\hat{\alpha}(m, n))$ تتطلبها العمليات على بنية معطيات المجموعات المنفصلة التي تحقق بعض الشروط التقنية. قام Fredman و Saks [113] فيما بعد بتعميم هذا الحد، وأظهرا أنَّه في أسوأ الحالات، يجب أن يجري النفاذ إلى كلمات في الذاكرة طولها $\Omega(m\,\hat{\alpha}(m, n))(\lg n)$ بتًّا.





مطبوعات الجمعية العلمية السورية للمعلوماتية

معجم مصطلحات المعلوماتية

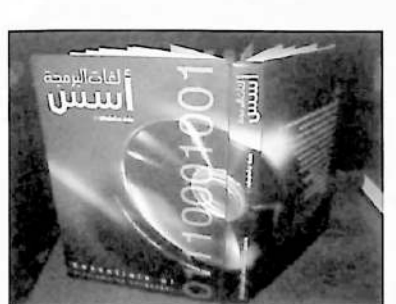
Dictionary of Information Technology Terms



- يضم 7000 مصطلح في شتى علوم المعلوماتية.
- يحوي المصطلح الأجنبي والمقابل بالعربي مع شرح للمصطلح باللغة العربية
- شارك في وضعه 30 باحثاً من هيئات علمية رفيعة.

أسس لغات البرمجة

Essentials of Programming Languages



- أدوات للبرمجة الرمزية — الاستقراء والعودية والمدى
- التجريد النحوي وتجريد المعطيات — قواعد الاختزال والبرمجة الأمرية
- المفسرات — تمرير المعاملات
- اللغات الغرضية التوجه — طراز تمرير الاستمرارات
- مفسرات تمرير الاستمرارات — الشكل الأمري وبناء المكدس
- الماسحات والمحللات النحوية — اشتقاق المترجمات

هندسة البرمجيات – المجلد الأول والمجلد الثاني

Software Engineering



- مقدمة – المنتج والإجرائية
- إدارة المشاريع البرمجية (مفاهيمها – مقاييس الإجرائية البرمجية والمشروع – تخطيط المشاريع البرمجية – إدارة المخاطرة – الجدولة الزمنية للمشروع – ضمان جودة البرمجيات – إدارة تشكيلة البرمجيات)
- طرائق تقليدية في هندسة البرمجيات (هندسة النظام – التحليل – التصميم – تصميم نظم الزمن الحقيقي – تقنيات اختبار البرمجيات – المقاييس التقنية للبرمجيات)
- هندسة البرمجيات الغرضية التوجه (مفاهيم ومبادئ – التحليل – التصميم – الاختبار – المقاييس التقنية)
- مواضيع متقدمة في هندسة البرمجيات (الطرائق الصورية – الغرفة النظيفة – إعادة استخدام البرمجيات – إعادة الهندسة – هندسة برمجيات الزبون/المخدم – هندسة البرمجيات بمعونة الحاسوب)

الذكاء الصنعي

Artificial Intelligence

- مدخل (ماهية الذكاء الصنعي – مناهج الذكاء الصنعي)
- الآلات التفاعلية – الشبكات العصبونية – ارتقاء الآلة – آلات الحالة – الرؤية الربوطية
- الوكلاء التي تخطط – البحث الأعمى – البحث التجريبي – التخطيط والفعل – البحث البديل – البحث المعاكس
- البحث في فضاءات الحالة (حساب الفرضيات – حساب الإسناديات – نظم قواعد المعرفة – تمثيل المعارف البديهية – المحاكمة باستخدام المعارف غير المؤكدة – التعلم والفعل باستعمال شبكات بايز)
- طرائق التخطيط المعتمدة على النطق (حساب الموقع – التخطيط)
- الاتصالات والتكامل (تعدد الوكلاء – الاتصال بين الوكلاء – بنيات الوكلاء)

مفاهيم نظام التشغيل – الجزء الأول والجزء الثاني

Operating System Concepts



- لمحة عامة إلى نظام التشغيل (مقدمة – بنية نظام الحاسوب – بنية نظام التشغيل)
- الإجرائيات – النياسب – جدولة وحدة المعالج – تزامن الإجرائيات – التوقف التام
- إدارة الخزن (إدارة الذاكرة – الذاكرة الافتراضية – نظام الملفات)
- نظم الإدخال والإخراج – بنية الخزن الوسيع
- النظم الموزعة – التنسيق الموزع
- الحماية والأمن
- دراسة حالات (نظام لينكس – نظام ويندوز 2000 – نظام ويندوز XP)
- نظام FreeBSD – نظام Mach – نظام Nachos

التعمية التطبيقية

Applied Cryptography

- أسس التعمية – لبنات بروتوكولات التعمية – البروتوكولات الأساسية – البروتوكولات المتوسطة – البروتوكولات المتقدمة – البروتوكولات الطلسمية
- تقنيات التعمية (طول المفتاح – إدارة المفاتيح – أنواع الخوارزميات وأنماطها – استخدام الخوارزميات)
- خوارزميات التعمية (مراجعة رياضية – مِقْيَس تعمية المعطيات DES – خوارزميات لبنية – معميات لبنية إضافية – ضم المعميات اللبنية – مولِّدات السلاسل شبه العشوائية والمعميات التسلسلية –مولدات السلاسل العشوائية الحقيقية – توابع البصمة الوحيدة الاتجاه – خوارزميات المفتاح العلني – خوارزميات التوقيع الرقمي بالمفتاح العلني – خوارزميات تبادل المفاتيح)

المدخل إلى Mathematica 5.0

Introduction To Mathematica 5.0

- لمحة تاريخية إلى Mathematica – ما هو Mathematica – المدى الواسع لاستخدامه



- بيئة العمل في Mathematica
- أساسيات في Mathematica
- البرمجة بلغة Mathematica 5.0
- تطبيقات Mathematica 5.0 في التحليل العددي (طريقة تنصيف المجال – طريقة القاطع – طريقة نيوتن-رافسن – طريقة النقطة الثابتية – طريقة هالي – طريقة مولر)

اتصالات المعطيات والحواسيب – المجلد الأول والمجلد الثاني

Data And Computer Communications

- تمهيد لمواضيع الكتاب (مواضيع حقل اتصالات المعطيات واتصالات الحواسيب – مفاهيم البروتوكولات وبنياتها
- تبادل المعطيات من نقطة إلى نقطة – تقانات النقل التماثلي والرقمي واللاسلكي – التحكم في الوصلات – التضميم.
- تبادل المعطيات وتقانات الاتصالات للشبكات الواسعة المدى (تقانات ابتدال الدارات والرزم و ATM والشبكات اللاسلكية الواسعة).
- التقانات والبنيانات للتشبيك على المسافات القصيرة – عناصر تصميم السبكات المحلية (أوساط الإرسال، والطبولوجيا، وبرتوكولات التحكم في النفاذ إلى الوسط) – دراسة لبعض الشبكات المحلية المقيَّسة.
- الآليات والمبادئ البنيانية اللازمة لتبادل المعطيات بين تجهيزات المعالجة المعطيات (حواسيب، محطات عمل، مخدمات) المرتبطة بالشبكات المحلية أو الشبكات الواسعة أو الشبكات البينية المؤلفة للإنترنت.

مسرد مصطلحات المعلوماتية

Glossary of Information Technology Terms



- يضم 5000 مصطلح جديد
- يحوي المصطلح الأجنبي والمقابل بالعربي
- يعد مكملاً لمعجم مصطلحات المعلوماتية

مجلة الثقافة المعلوماتية



- مجلة تخصصية تعنى بالبحوث الحديثة المنشورة في أرقى الدوريات العالمية في المعلوماتية، وتهتم بالترجمة الدقيقة لهذه البحوث مع المراجعة العلمية والضبط اللغوي. يصدر أربعة أعداد منها كل سنة.
- صدر منها حتى الآن 40 عدداً

تطلب جميع مطبوعات الجمعية من المقر الرئيسي للجمعية (مجلس إدارة الجمعية)
ومن جميع فروع الجمعية في المحافظات (اللجان الإدارية)

للاستعلام

هاتف : 3736156 - 2137204 - 2137205

جوال : 0933545981 – 0932503964

للمراسلة

دمشق – الجمارك – بجانب وزارة التعليم العالي

ص.ب. 33492

فاكس : 3737558 – 2137202

بريد إلكتروني: nzhafez@scs-net.org

